68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $ 9.45 | Hong Kong H $23.50
Malaysia M $ 9.45 | Sweden 30:-SEK

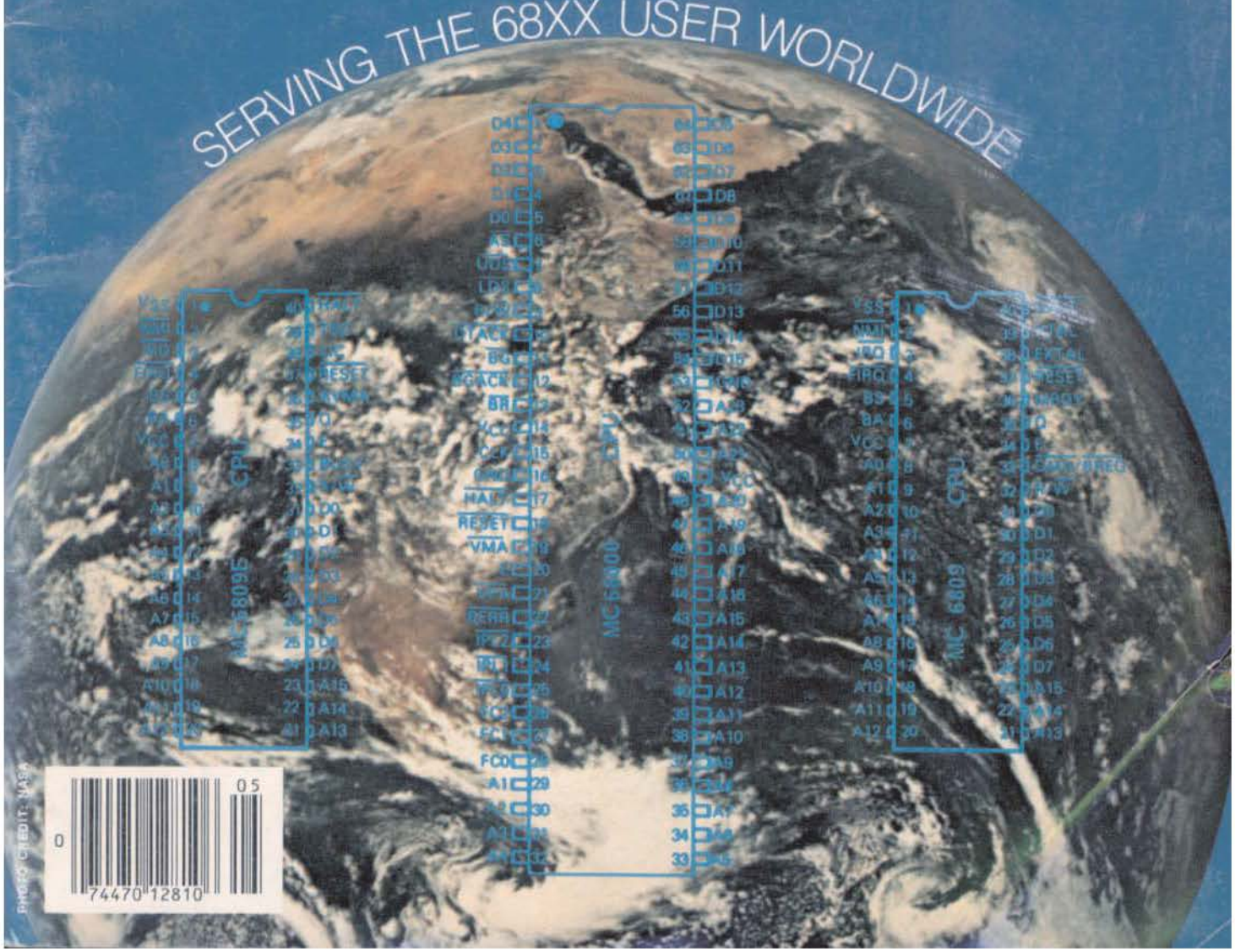
$2.95 USA

68020

6809

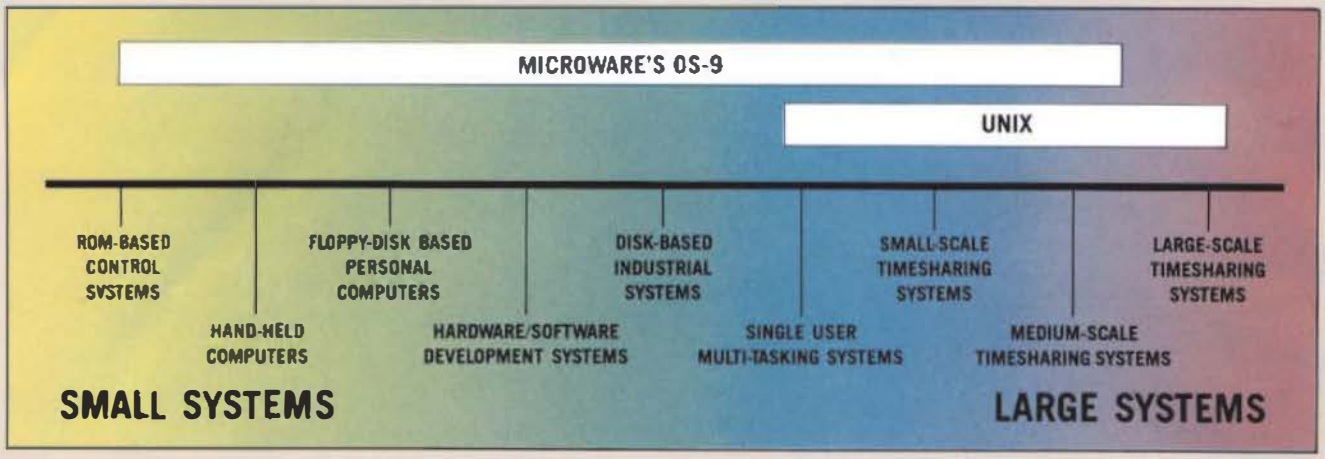
VOLUME VIII ISSUE V • Devoted to the 68XX User • May 1986

"Small Computers Doing Big Things"



0 74470 12810 05

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
5:1 Pengatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL GMBH
Porphyrstrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DSDD Floppies, Cabinet & Power Supply $1898.88
60MB Streamer (UniFLEX-020 only) $2400.00
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) $349.67
#64 Static RAM-64K CMOS w/battery (6809 Only) $396.64
#72 256K CMOS Static RAM w/battery $998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.48
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS — SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $38.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS

GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMXI Included	GMX II Included	GMX III Included			
FLEX	Included	Included	Included			
GMXBUG Monitor	Included	Included	Included			
Basic 09, Run8 (OS9)	Included	Included	Included			
RMS (OS9)	Included	Included	Included			
DO (OS9)	Included	Included	Included			
VDisk for FLEX	N/A	Included	Included			
RAMDisk for OS9	N/A	$125 option	Included			
O-FLEX	N/A	$250 option	Included			
Support ROM	N/A	N/A	Included			
Hardware CRC	N/A	N/A	Included			

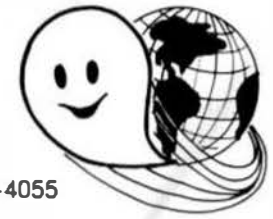
Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX inc.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055



'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS – HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S,9 - 5/8 DMF Disk - CDS1 - 8212W - Sprint3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Contents

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year.** Must be in **U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH **or** TSC Editor format with **3.5 inch column width.** All disks will be returned. Articles submitted **on paper** should be **4.5 inches in width** (Including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: **.sp** space, **.pp** paragraph, **.fi** fill and **.nf** no fill. Also please **do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **muat be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG -020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the **REAL KICK!**"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be Turtle or Thoroughbred?

MUSTANG-020 System Prices
12.5 Mhz

Mustang-020 SBC, wired & tested with 4 D825 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives.....$2750.00

M020 Cabinet and P/S, for Mustang-020, less cables..... $299.95

M020 Cables, dual floppy or HD specify which floppy or winchester.....$39.95

M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies.....$79.95

M020F Floppy, 80 track, DD/DS.....$269.95

OS-9, SPECIAL Mustang-020 version.....$350.00

MC6881 F/P co-processor.....$495.00

20 Megabyte Winchester.....$895.00

Winchester HD Controller.XEBEC.....$395.00

8 Port Expansion (complete).....$495.00

** Special Winchester Notice **

The Mustang-020 device descriptors will allow you to use practically ANY winchester drive supported by XEBEC 1410/1410A or OMTI 20C-1 controllers.

. . .

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H. Complete System include $20.00. All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

Dealer & Quantity Discounts Available

* Mustang-020 is trademark of Data-Comp-CPI

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
TELEX 5106006630

MUSTANG-020 Benchmarks **
Time Seconds

Type System	32 bit Int. Loop	Register Long Loop
IBM AT 7300 Xenix Sys 3	9.7	No Registers
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " " "	5.1	3.2
68008 OS9 68K 8 Mhz	18.0	9.0
68000 " " 10 Mhz	6.5	4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UNIFLEX "	1.8	1.22

** Loop: Main()
{
register long i;
for (i=0; i < 999999; ++i);
}

Estimated MIPS - MUSTANG-020 - 2.5 MIPS
Motorola Specs: Burst up to 7 - 8 MIPS - 16 Mhz

For a limited time we will offer $400 Trade-In on gour old Q--- 68008 or 68000 SBC, must be working properly and complete with all software and cables. Call for more information!

Mustang-020 Software

OS-9

OS-9	$350.00
Basic09	300.00
C Compiler	400.00
Fortran 77	*400.00
Pascal Compiler	400.00
OMEGASOFT-PASCAL	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
PAT	229.00
JUST	79.95
PAT/JUST Combo	249.50
SCULPTOR+	**995.00
COM	125.00

** See discount below

UniFLEX

UniFLEX	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM.w/source	100.00
TMODEM.w/source	100.00
X-TALK.see.adv	99.95
Cross Assembler	50.00
Fortran 77	450.00
SCULPTOR+	**995.00

** See discount below

* New items

Standard system shipped 12.5 MHz
Add for 16 Mhz..68020....$400.00
Add for 16 Mhz..68881....$400.00
8 Port expansion board use two total of 20 RS232 ports wired & Tested.....each....$498.00

SCULPTOR+..We are USA distributors for SCULPTOR+. Call or write for site license or multiple discounts.
SCULPTOR discount with complete system - Special - $695.00. OS-9 or UniFLEX.

** Software Discounts
Call for software discounts from 10-70% for buyers of these systems, from Data-Comp. Limited offer.
Call!

Mustang 020 Features

* 12.5 MHz (16.6 MHz optional) MC68020 full 32-Bit wide Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatible with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
* Optional 68881 Floating point Coprocessor (12.5 MHz or 16.6 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
* 2 Megabytes of RAM (512K x 32-bit organization)
* Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
* 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
* 8 RS-232 Expansion ports (max. 20)
* Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
* Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
* Time-of-Day Clock/Calendar w/battery backup
* Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
* Mounts Directly to a Standard 5 1/4" Disk Drive
* SASI Interface for Intelligent Hard Disk Controllers
* Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
* 5-bit sense switch, readable by the processor
* Hardware single-step capability

** ACTION PROVEN **

The MUSTANG-020 is already on the job! And winning acclaim in industry, commerce, business and several government agencies. The delivery times were close to schedule. We are hearing back nothing but praise (and more orders).

If you are considering the purchase of a Mustang-020, be advised that the price will increase in the second quarter of this year.

Experienced users are awed at the tremendous power and speed of the Mustang-020, from Data-Comp. Especially when compared with other 68XXX systems. Not only is it more practical than all the others, but it is much more cost efficient. Compare it to any other 68XXX and you will see why!

Dual 5" 80 trk. Floppy No Winchester		Winchester & 1 Floppy		
020 Board	$2750.00			$2750.00
Cabinet	299.95			299.95
5"-80 trk Floppy(2)	539.90		(1)	269.95
Floppy cable	39.95			39.95
OS-9 68K	350.00			350.00
		Winchester cable		39.95
Total System	$3979.80	Winchester controller		395.00
Less 5%	-198.99	25 Mbyte Winchester		895.00
	$3780.81	Total System		$5039.80
S/H UPS	20.00	Less 5%		-251.99
Total	$3800.81			$4787.81
		S/H UPS		20.00
		Total		$4807.81

NOTE: 68881 Co-Processor Add $495.00
UniFLEX $450.00
less $350.00 (OS-9) Add $100.00

Prices and Specifications subject to change.

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

A few months ago, I presented a program that opens an input and an output file and simply reads the input file and writes it to the output file. I did that as an example of file handling in FLEX. I also pointed out that while the file is running through the processor you can "filter" it. That is, you can make changes in the string of characters to do things that you want to do to convert a file from one format to another, for example.

This month I have prepared two useful utility programs in Assembler based on that skeleton program. The first and simpler of the two, reads a text file in Stylograph format and writes an output file in PAT or EDIT format. The difference is fairly small, but very important. Stylo puts no carriage returns within a paragraph. It saves each paragraph as one very long line. Most other editors including PAT, PIE, and TSC EDIT expect lines not to exceed 128 bytes or so, and simply throw away the overflow when reading a STYLO file. This utility called STYTOPAT, simply reads the file into a line buffer. If a CR is detected, it writes a line to the output file. If the line gets too long, it changes the next space to a CR and writes the line to the output file.

The only complication beyond our skeleton program to copy a file, is that it reads the input file into the line buffer and then writes a line to the output file. The section of code from line 58 to line 81 do all the conversion work. I have commented nearly every line of that section. Essentially the X register must be set to point at the input file File Control Block, so I used the Y register to point at the line buffer position where the next character is to be placed. At the start of RDLOOP, a counter RCOUNT is set to zero. It is incremented for each character stored in the line buffer. A few lines of code look for spaces at the start of a line and throw them away. If this is not done, double spaces at the end of sentances put a space at the start of the next line, should the line happen to break at the end of a sentence.

When the RCOUNT gets to be 60 (the constant LENGTH at the start of the program) the input file is read until a space is found. The space is overwritten with a CR and the execution falls through to the label WRITE. WRITE points X at the output file FCB, clears a WCOUNT, and enters a loop to read the line buffer using Y as the pointer. It increments WCOUNT for each character written, and when WCOUNT equals RCOUNT, it is done and branches around to RDLOOP to get another line. End of file causes exit as in the sample program given previously.

BYTES is a program that counts the number of bytes in a binary file and reports in both HEX and Decimal form. It also reports the program starting or transfer address. I first did BYTES in PL/9 and I will include that listing so you can compare the programs. I note that having done the PL/9 program I simply went down the program and coded it in Assembler almost line for line, except that I was able to use a lot of FLEX routines in the process. BYTES only reads an input file. There is no output file, but rather, it reports the total bytes and transfer address on the terminal. I therefore had only to open the input file. Perhaps this is a good point to break and discuss the FLEX routines used. These are all described in the Advanced Programmer's Guide, which many of you don't have.

I have prepared two useful utility programs in Assembler based on a skeleton program that opens an input file and simply reads the input file and writes it to the output file....

BYTES is a program that counts the number of bytes in a binary file and reports in both HEX and Decimal form.

It also reports the program starting or transfer address.

PUTCHR ($CD18)

This routine expects you to have put a character in the A accumulator. It outputs that character to the terminal and does not change the contents of any register except the A accumulator. All of these FLEX routines are entered with a simple JSR to the address indicated.

PCRLF ($CD24)

This routine outputs a Carriage Return and Linefeed to the terminal. PCRLF counts the lines that you have output and causes the FLEX PAUSE feature to work after you have output the number of lines you have set with the TTYSET utility if PAUSE is enabled. You need not load any register with anything in particular before calling PCRLF, and no registers contents are lost.

OUTADR ($CD45)

This routine outputs four Hexadecimal digits to the terminal load X with the address of a 16 bit integer value (for example T0TBYT in the BYTES program) and JSR OUTADR, and the Hexadecimal value will be output to the terminal.

OUTDEC ($CD39)

This routine outputs a 16 bit value as a decimal number in the range -32768 to +32767. If the B accumulator contains zero, it simply outputs the number using the number of character positions that the number occupies. If B is not zero, it includes leading spaces to make the total character positions 6 (five for the number and one for a minus sign).

FMS ($D406)

This is the File Management System call. What it does depends on the function code that is set up in the File Control Block to which X is pointing. For example if X points at INFILE in STYTOPAT, and the file is open for READ, FMS returns the next character of the file in the A Accumulator. If no error is detected, the zero flag of the processor is set. If an error IS detected the zero flag is clear. BNE ERROR following the FMS routine call will send the program operation to the ERROR handler if an error has been detected. ERROR #8 is the signal for end of input file, and the program exits normally when that error is detected. All other errors trigger a call to the next FLEX routine

RPTERR ($CD3F)

RPTERR reads the error status byte in the FCB. The X register must be pointing at the FCB when the call is made. RPTERR reads the FLEX ERRORS_SYS file and reports the appropriate file error message and returns to the program.

GETFIL ($CD2D)

On entry, the X register points to an FCB. This routine gets the filename from the command line and puts it in the FCB.

SETEXT ($CD33)

This routine is entered after GETFIL. If the user did NOT specify an extension with the filename on the command line, SETEXT will supply a default extension specified by the program. An extension code must be in the A accumulator. If the user DID specify an extension, it will be left alone. A table of the codes follows:

0 BIN
1 TXT

FMSCLS ($D403)

This is the "emergency bail out" routine. If a file error is detected, this routine may be used to close all open files immediately.

WARMS ($CD03)

All programs that are run under FLEX should normally exit by a jump to this address. WARMS restarts FLEX so that the +++ prompt appears and it is ready to accept another command.

I realize that these descriptions are brief, but a look at the program listings will show how they are used. Now to continue discussing the program BYTES, we must first describe a binary file in FLEX. The first byte of a binary file is always $02, which is the signal for a block of bytes to be loaded. It is followed by a double byte that is the load address. After that comes a byte count indicating how many bytes are in the block to be loaded. Since a single byte can represent numbers up to 255 when considered as an unsigned value, the maximum number of characters in a block is 255. When the block has been loaded, the next character must be either another $02 designating another block to be loaded, or a $16 indicating that the next double byte is a transfer address (starting address for the program). Though a program might have more than one transfer address, the last one is the one that FLEX uses, so we simply save the transfer address in XFERAD for printing at the end of the program.

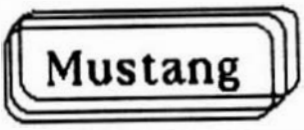
This would all be quite simple except for one thing. Some of the compilers including PL/9 that are single pass compilers, write nulls to the output file if the destination of a jump or branch is not known when the output file is written. Later, they overwrite the branch address with the correct value by means of setting the load address back to the jump location and loading a few bytes there. If we count all the overlays, the byte count for the program is incorrect. Logic has therefore been built in to see that a load address is earlier than the last loaded byte, and those bytecounts are not added to the total. After the file is closed, the values accumulated as the total byte count are output using OUTADR and OUTDEC, in Hexadecimal and Decimal form.

You can see that the assembler program listing is about twice as long as the PL/9 listing. The PL/9 program has one problem. Integers in PL/9 are SIGNED, so that if the load range of the program happens to cross $8000, the wrong byte count is given. The program could be written in "C" since that language has an "unsigned int" variable type. Once the assembler program was working I made a few improvements in it. If the first byte of the file is not $02, the program exits and reports "FILE NOT BINARY".

Both of these BYTES programs could be fooled by a program that, for example, first overlays a FLEX jump vector in high memory and then loads the program in low memory. If you write utilities to do this, and want a byte count, you will have to put the overlays at the end of the program so that the load address will not go backwards. It would be possible to write a considerably more complex BYTES program that would maintain a load map of all program bytes loaded, and count all bytes not within the previously loaded areas, whether forward or backward in load address.

I hope these programs might serve two or three good purposes for those of you who are interested in writing assembler programs. First, they illustrate how the FLEX routines are used. Secondly, the BYTES program follows the PL/9 program rather closely. It might serve to show how to code a program in assembler once you have a high level language version of it working. I have often done a program in BASIC first, and then used that listing for a model for coding of the Assembler program, I note that the Assembler version is about 325 bytes long, and the PL/9 version is about 825 bytes. I also would like to

point out that the Assembler program would be much longer (both listing and total code) if it did not use the FLEX routines such as OUTDEC and OUTADR. The same may be said for the PL/9 program listing, although the code for the I/O library functions IS included in the byte count for that program.



Before anyone gets all bent out of shape, I know, this is a FLEX column (though it may undergo a name change in the near future). This is just so interesting and important, that I must give you the low down. My company ordered a MUSTANG-020 from South East Media in December. It has arrived. Remember now, that I have never operated a computer with OS-9 before (except for a few evenings with it on the Color Computer some time back). On first glance, the whole thing looked rather overwhelming. The box reached my office about 3:00 one afternoon. I spent some time with my co-worker Doug Case looking at it and trying to decipher what surely could have been Greek just as easily as not. Next morning Doug was ill and I had the computer all to myself. By 9:00 I was booting from the hard disk. That afternoon I loaded Microware "C", and compiled my first C program on it. It took 21 seconds to compile a "Hello There" program. I must admit that I am not used to OS-9, and I had not LOADed the C compiler modules into RAM first.

Later I decided to get the Ram Disk enabled, and put all the C library files on it. Then I moved my source file to RAM Disk and that same program compiled in 4.7 seconds flat. Of course that is with the C compiler "loaded" into memory also. For a comparison, Lattice "C" on the Tandy 1200-HD (8086 processor) compiled that program in 41 seconds. My 6809 system running at 2 MHz with an 8" double density floppy disk compiled the "hello there" program in 103 seconds.

Being a one-track person, as I am, I tucked the system under my arm and brought it home. Let me preface the following remarks by saying that Windrush McCosh "C" and Microware "C" have the same origin, both having been written by James McCosh. Microware purchased that C and has done a lot of things with it, including porting it to the 68000 environment. What I mean is that the difference in execution times reported below HAS TO BE primarily the difference in processors and hardware (Ram Disk and pre-loaded programs).

A couple of years ago I wrote a benchmark program with which I have tested a number of compilers for compile time and execution time doing a number of different sorts of calculations and data movements. I had to multiply the loop counts for the various tests by ten so I could time the 68000. The times have been divided by ten to reflect the same loop counts as the 6809 benchmark. The 6809 times reflect those of a 2 MHz. 6809 system running FLEX

	6809	68020
Compile time	120 sec.	8.7 sec.
Array Assign	4	0.62
Square Root	14	1.00
Integer * /	3.2	0.45
Real Mul Div	18.5	1.18
Param Passing	2.75	0.93
Sieve	10.0	1.33
Bytes	11K	16K

As a matter of interest, the total compiler size (in Bytes) is roughly 110K for the 6809 version and 155K for the 68000 version. It is interesting to note that the compiler size ratio is almost the same as the output code size ratio. The compile times above certainly reflect the fact that one was done with floppy disks and the other primarily in RAM. I think this is a fair comparison because Ramdisk is not available to many of us that have 6809 systems.

The Array Assign test assigns integer values to all the elements of an array of dimension 1000, and it repeats that operation 100 times. The square root of 1000.0 is taken 100 times to a precision of 7 digits. There are 8000 integer multiplies and divides in the next test. The Real multiply divide test is repeated 2000 times. The next test passes two Real parameters to a procedure which assigns them to two local variables and returns. This procedure is repeated 10000 times. The value for Bytes in the table is the output code size. Remember that the 68020 uses machine instructions that are in general twice as large as the 6809 instructions. It wouldn't be extremely unreasonable to expect the 68000 compiler to generate twice as much code. However, the 68000 instructions do more. They operate on a wider data bus. The math (particularly the Real math) can be done with less instructions AND fewer passes through loops.

As you can see, the real winner (no pun intended) is the Real math operations, which run nearly 18 times faster on the 68020 than on the 6809. Needless to say, I am impressed. It would appear that the progress is real and not imaginary. As I write this, I have this little box on the dining room table. It is smaller than my two 8" disk drives. It has over 32 times the memory of my 6809 system. The hard disk holds as much as about 22 of my DSDD 8" floppy disks. It runs an average of ten times faster in most operations. If I were running the 6809 at 1 MHz, I would be even more impressed!!

Further Impressions

I am writing this about a week after the above. In that week or a little more, I have managed to get JUST running on the MUSTANG-020 (the C version of course), and to translate my editor PAT into C and get it running on the MUSTANG-020 as well. Presently it all works but I have a few rough edges to polish before I start to distribute copies for testing. I don't think the debug will be as long as it was for the original FLEX version, since the debug was largely a matter of finding the places where my translated code didn't do what I expected. I learned one lesson about C that I will heed from now on. When in doubt about operator precedence, use parentheses. For example, in one place I had the expression k= cury << 7 + curx; That is supposed to calculate the array index for a position on the screen represented by line cury and column curx. It seems that the addition takes precedence over the shift operator so what I got was: k= cury << (7+curx). What I wanted was: k= (cury << 7) + curx. Of course the way C evaluated it gave me nonsense that took a couple of days to sort out.

Now for some further impressions of the 68020 system and OS-9. First let me say that the system has performed flawlessly. One day I reached over and accidentally turned the system off while writing a file. That caused me to have to reformat the hard disk. Such stupidity on my part! Other than that, the system has been completely reliable. I learned how to use the screen editor that comes with OS-9 (called SCRED) and found it quite reasonable to use to get PAT typed in and debugged to the point where I could start using it on itself. It is almost impossible to separate the impressions of the hardware from the actions of the software, so perhaps the two will merge unavoidably here.

OS-9 is not difficult to learn, except for one problem I had because I had been using an IBM clone for the past year. MS-DOS uses the command 'cd' to change directories. OS-9 uses 'chd'. It took about three days for me to make the switch. In that time I saw an error message probably 20 times an hour. That is not a complaint about OS-9, since once I got used to the command all went well. OS-9 DOES a lot of things. By that I mean that even when you are running a program, it still monitors a number of keys on the terminal. For example ^C will interrupt the program and ^E will abort it. ^S (xoff) will stop output to the terminal until you type ^Q (xon). This is all nice except that PAT uses all those keys for editing functions. It turns out that there is no problem because OS-9 provides a way to disable those functions through a utility called tmode (not unlike TTYSET in FLEX). Fortunately there is provision for system calls in Microware C and it is possible to call tmode to set all the key values to nulls, thus disabling those functions from the point of view of the operating system. PAT restores the "normal" tmode parameters on exit.

A day or two after the MUSTANG-020 arrived, we got curious and connected a second terminal. We found it simple to get it going, and noted that two people could use it with no degradation in performance noted except when compiling a large program. Of course the time the processor is available is divided between the users, and it was obvious that there was another terminal connected. In operations like multi-user word processing systems, a user would never notice any degradation in performance. I'd guess the little box would support 8 or 10 users in such applications quite nicely. We found that we could each list a file to our terminals simultaneously and not notice when the other user finished listing. That is, the system can handle two terminals at 19.2K baud without slowing either one down.

This discussion wouldn't be complete without mention of the fact of the 2 megabytes of RAM. OS-9 nicely uses a part of that memory for a RAMdisk, which greatly speeds up the compile operation. I was able, with the RAMdisk functioning and all of the C compiler LOADed into RAM, to provide for PAT to use an edit buffer of 100K! All of the PAT C source file fits in less than half of that edit buffer. It is a great convenience to be able to edit very large files in one chunk just for the convenience of global changes. I could run back and forth from beginning, to look at variable declarations to end, to add debug functions or to check on some of the lowest level functions, over and over again without spooling some of the file to disk. Of course the buffer could be much larger, but I feel 100K to be more than adequate. SCRED allows the user to specify the buffer size, but has a default of only 16K, which is good enough for about 4 pages of single spaced typed text, too small for my tastes. I got tired of typing in: SCRED PAT.C -b=70k or its more stripped down version -b70. I'd rather have a configurable default, or one that is large enough for all practical editing.

Lastly, I want to say that these words are unsolicited. I was NOT given a Mustang-020 to play with. The company for which I work BOUGHT one at the advertised price. We BOUGHT OS-9 and the Microware C compiler. My good review here is based on the PERFORMANCE of the system indicated above in the benchmark tests, and as "felt" during the intensive use of it for a week. Perhaps I should report that PAT 6809 version in PL/9 is now 16,700 bytes more or less. PAT in Microware C on the Mustang-020 is 22,700 bytes! I had fully expected it to grow to 32K or so, and was very pleasantly surprised at its final size. Best of all, with the C compiler LOADed into RAM and the .LIB files on the ramdisk, All of PAT compiles in 75 seconds flat. Were it not for this performance, it would have taken me several times as long to get PAT translated and debugged. I should also mention that we did not buy the math co-processor for the system, so all the timing is based on the 68020 alone. My hat is off for the fine single board computer, to Data Comp for the nice packaging of this system, and to Microware for a very good operating system. The Mustang-020 looks nice, is small enough to carry under one arm (about like a large dictionary) and VERY capable. I am also impressed with the large amount of software already available for it, and the other software coming along soon.

I should mention that OmegaSoft Pascal is already available for it. I have a copy ready to test, and I should have a review available by next month. I also have a copy of Stylo for it, which will be reviewed shortly.

- - -

```
                        *
                        *
                        * THIS PROGRAM CONVERTS A STYLO TEXT FILE TO
                        * A PAT TEXT FILE BY INSERTING CR'S
                        *
                        * EQUATES FOR FLEX ROUTINES
                        *
              CD18  PUTCHR  EQU   $CD18
              D406  FMS     EQU   $D406
              D403  FMSCLS  EQU   $D403
              CD20  GETFIL  EQU   $CD20
              CD3F  RPTERR  EQU   $CD3F
              CD33  SETEXT  EQU   $CD33
              CD03  WARMS   EQU   $CD03
              003C  LENGTH  EQU   60      MAX LINE LENGTH FOR WRITE TO OUTPUT FILE
                        *
0000                    ORG   0
                        *
                        * GET INPUT FILE NAME
                        *
0000 8E 003A    START   LDX   #INFILE   POINT X AT THE FILE CTRL BLOCK
0003 BD CD20            JSR   GETFIL    GET FILENAM FROM COMMAND LINE TO FCB
0006 1025 0096          LBCS  ERROR     ERROR IF RETURNS WITH CARRY FLAG SET
000A 86 01              LDA   #1        DEFAULT EXT .TXT
000C B9 CD33            JSR   SETEXT    SET EXTENSION IF ONE WAS NOT SUPPLIED
                        *
                        * GET OUTPUT FILE NAME
                        *
000F 8E 0FFA            LDX   #OUTFIL   POINT AT OUTPUT FILE CTRL BLOCK
0012 BD CD20            JSR   GETFIL    GET FILENAME FROM COMMAND LINE
0015 25 79              BCS   ERROR
0017 86 01              LDA   #1        DEFAULT EXTENSION .TXT AGAIN
0019 BD CD33            JSR   SETEXT
                        *
                        * OPEN INPUT FILE FOR READ
                        *
001C 8E 003A            LDX   #INFILE   POINT AT THE FCB AS USUAL
001F 86 01              LDA   #1        FUNCTION CODE FOR "READ"
0021 A7 84              STA   0,X       FIRST BYTE OF FCB
0023 BD D406            JSR   FMS       OPEN FOR READ
0026 26 6B              BNE   ERROR
                        *
                        * OPEN OUTPUT FILE FOR WRITE
                        *
0028 8E 0FFA            LDX   #OUTFIL   POINT AT FCB AGAIN
002B 86 02              LDA   #2        FUNCTION CODE FOR WRITE
002D A7 84              STA   0,X       FIRST BYTE OF FCB
002F BD D406            JSR   FMS       OPEN FOR WRITE
0032 26 5C              BNE   ERROR
                        *
                        * NOW READ CHARACTERS FROM INPUT FILE
                        * AND PUT THEM INTO A LINE BUFFER
                        *
0034 8E 0054    RDLOOP  LDX   #INFILE   START OF READ WRITE LOOP
0037 108E 033A          LDY   #LINE     START AT BEGINNING OF LINE BUFFER
003B 4F                 CLRA
003C B7 0309            STA   RCOUNT    CHARACTER COUNT = 0
003F            RDLP1
003F BD D406            JSR   FMS       GET A CHARACTER
0042 26 4C              BNE   ERROR
0044 84 7F              ANDA  #$7F      REMOVE PARITY BIT ALWAYS FOR ASCII TEXT
0046 7D 0309            TST   RCOUNT
0049 26 04              BNE   RDLP2     SKIP SPACES AT START OF A LINE
004B 81 20              CMPA  #$20      IS IT A SPACE
004D 27 FB              BEQ   RDLP1     DON'T SAVE LEADING SPACE
004F A7 A0      RDLP2   STA   ,Y+       PUT CHAR IN LINE BUFFER
```

```
0051 F6 0389          LDB   RCOUNT
0054 5C               INCB
0055 F7 0389          STB   RCOUNT    INCREMENT COUNT
0058 81 0D            CMPA  #$0D      IS IT CR
005A 27 0C            BEQ   WRITE     IF SO WRITE LINE TO OUTPUT FILE
005C C1 3C            CMPB  #LENGTH   IS LINE FULL?
005E 2F 0F            BLE   RDLP1     IF NOT GET MORE
0060 81 20            CMPA  #$20      IF CURRENT CHAR NOT SPACE, GET MORE
0062 26 09            BNE   RDLP1
0064 86 0D            LDA   #$0D      IF CURRENT CHAR SPACE, MAKE IT CR
0066 A7 1F            STA   -1,X      OVERWRITE SPACE
                      *
                      * HAVE LINE IN LINEBUFFER
                      * NOW WRITE TO OUTFILE
                      *
0068 108E 033A WRITE  LDY   #LINE
006C 8E 01FA          LDX   #OUTFIL   POINT AT OUTFILE FCB
006F 4F               CLRA
0070 B7 038A          STA   WCOUNT    WRITE CHARACTER COUNT
0073 A6 AB    WRLOOP  LDA   ,Y+
0075 BD D406          JSR   FMS       WRITE TO OUTPUT FILE
0078 26 16            BNE   ERROR
007A BD CD18          JSR   PUTCHR    PUT ON SCREEN FOR VERIFICATION
007D B6 038A          LDA   WCOUNT
0080 4C               INCA
0081 B7 0384          STA   WCOUNT
0084 81 0389          CMPA  RCOUNT    OUTPUT WHOLE LINE?
0087 2D EA            BLT   WRLOOP    IF NOT GET MORE
0089 86 0A            LDA   #$0A
008B B8 CD18          JSR   PUTCHR    NEW LINE TO TERMINAL CALL
008E 20 66            BRA   RDLOOP    GO AROUND AGAIN UNTIL ERROR OR EOF
                      *
                      * ERROR HANDLER
                      *
                      * PROGRAM MUST EXIT VIA EPROP SINCE END OF FILE
                      * IS DETECTED BY PRESENCE OF ERROR CODE
                      *
0090 A6 01    ERROR   LDA   1,X       GET ERROR CODE
0092 81 08            CMPA  #0        IS IT END OF FILE?
0094 27 09            BEQ   EXIT      IF YES, PROGRAM COMPLETED SUCCESSFULLY
0096 BD CD3F          JSR   RPTERR    ELSE REPORT ERROR AND RETURN TO FLEX

0099 BD C403          JSR   FMSCLS    QUICK CLOSE OF ALL OPEN FILES
009C 7E CD03          JMP   WARMS     ERROR EXIT
                      *
                      * NORMAL EXIT WITH CLOSE OF OPEN FILES
                      *
009F 8E 008A  EXIT    LDX   #INFILE   POINT AT THE FCB
00A2 86 04            LDA   #4        CODE FOR CLOSE FILE
00A4 A7 84            STA   0,X
00A6 BD D406          JSR   FMS       NORMAL CLOSE OF INFILE
00A9 26 E5            BNE   ERROR     YOU CAN HAVE AN ERROR CLOSING A FILE TOO
00AB 8E 01FA          LDX   #OUTFIL   NOW CLOSE THE OUTPUT FILE
00AE B6 04            LDA   #4        CODE FOR CLOSE FILE
00B0 A7 84            STA   0,X
00B2 BD D406          JSR   FMS       NORMAL CLOSE OF OUTFIL
00B5 26 09            BNE   ERROR
00B7 7E CD03          JMP   WARMS     ALL DONE, BACK TO FLEX
                      *
                      * FILE CONTROL BLOCK ALLOCATION
                      * IMMEDIATELY FOLLOWING PROGRAM CODE
                      *
008A                  INFILE RMB  32D     INPUT FILE CONTROL BLOCK
01FA                  OUTFIL RMB  32D     OUTPUT FILE CONTROL BLOCK
033A                  LINE   RMB  127
0389                  RCOUNT RMB  1
038A                  WCOUNT RMB  1
                      *
                             END  START

0 ERROR(S) DETECTED


SYMBOL TABLE:

ERROR  0090  EXIT   009F  FMS    D406  FMSCLS D403  GETFIL CD2D
INFILE 008A  LENGTH 003C  LINE   033A  OUTFIL 01FA  PUTCHR CD18
RCOUNT 0389  RDLOOP 0034  RDLP1  003F  RDLP2  004F  RPTERR CD3F
SETEXT CD33  START  0000  WARMS  CD03  WCOUNT 038A  WRITE  0065
WRLOOP 0073
```

```
                      *
                      *
                      * THIS PROGRAM COUNTS BYTES IN A BINARY FILE.
                      *
                      * EQUATES FOR FLEX ROUTINES
                      *
CD24                  PCRLF  EQU  $CD24
CD45                  OUTADR EQU  $CD45
CD39                  OUTDEC EQU  $CD39
CD18                  PUTCHR EQU  $CD18
D406                  FMS    EQU  $D406
D403                  FMSCLS EQU  $D403
CD2D                  GETFIL EQU  $CD2D
CD3F                  RPTERR EQU  $CD3F
CD33                  SETEXT EQU  $CD33
CD03                  WARMS  EQU  $CD03
                      *
C100                  ORG    $C100    FLEX UTILITY SPACE
                      *
                      * GET INPUT FILE NAME
                      *
C100 8E C26A  START   LDX   #INFILE   POINT X AT THE FILE CTRL BLOCK
C103 BD CD2D          JSR   GETFIL    GET FILENAME FROM COMMAND LINE TO FCB
C106 1025 0BB4        LBCS  ERROR     ERROR IF RETURNS WITH CARRY FLAG SET
C10A 86 00            LDA   #0        DEFAULT EXT .BIN
C10C BD CD33          JSR   SETEXT    SET EXTENSION IF ONE WAS NOT SUPPLIED
                      *
                      *
                      * OPEN INPUT FILE FOR READ
                      *
C10F 8E C26A          LDX   #INFILE   POINT AT THE FCB AS USUAL
C112 86 01            LDA   #1        FUNCTION CODE FOR "READ"
C114 A7 84            STA   0,X       FIRST BYTE OF FCB
C116 BD D406          JSR   FMS       OPEN FOR READ
C119 26 73            BNE   ERROR
C11B 86 FF            LDA   #$FF
C11D A7 88 3B         STA   59,X      SET BINARY FILE FLAG
                      *
                      * NOW READ CHAR FROM INPUT FILE
                      *
C120 8E C26A  RDLOOP  LDX   #INFILE   START OF READ WRITE LOOP
C123          RDLP1
C123 BD D406          JSR   FMS
C126 26 66            BNE   ERROR
C128 81 16            CMPA  #$16      TRANSFER ADDRESS?
C12A 26 10            BNE   RDLP2
C12C BD D406          JSR   FMS       IF SO GET AND SAVE IT
C12F 26 5C            BNE   ERROR
C131 B7 C3AF          STA   JFERAD    HI BYTE
C134 BD D406          JSR   FMS
C137 26 55            BNE   ERROR
C139 B7 C3B0          STA   JFERAD+1  LO BYTE
C13C 39               RTS
C13D 7D C3B3  RDLP2   TST   FIRST

C140 27 09            BEQ   RDLP4
C142 81 02            CMPA  #2
C144 1026 0BA5        LBNE  NOTBIN    IF NEITHER IT IS A TEXT FILE
C148 7F C3B3          CLR   FIRST
C14B BD D406  RDLP4   JSR   FMS       GET LOAD ADDRESS
C14E 26 3F            BNE   ERROR
C150 B7 C3A8          STA   CURLOD    HI BYTE
C153 BD D406          JSR   FMS
C156 26 37            BNE   ERROR
C158 B7 C3AC          STA   CURLOD+1  LO BYTE
C15B BD D406          JSR   FMS       GET LOAD BYTE COUNT
C15E 26 2F            BNE   ERROR
C160 B7 C3AA          STA   BYTECT
C163 FC C3A9          LDD   CURLOD    CURRENT LOAD ADDRESS
C166 10B3 C3AD        CMPD  OLDLOD    PREVIOUS LOAD ADDRESS
C16A 25 17            BLO   RDLP3     IF OVERLAY DON'T COUNT IT
C16C F6 C3AA          LDB   BYTECT
C16F 4F               CLRA
C170 F3 C381          ADDD  TOTBYT    ADD BYTECOUNT TO TOTAL BYTES
C173 FD C381          STD   TOTBYT    ACCUMULATE TOTAL
C176 BE C26A          LDX   #INFILE
C179 F6 C3AA          LDB   BYTECT    UPDATE OLD LOAD ADDRESS
C17C 4F               CLRA
C17D F3 C3A8          ADDD  CURLOD
C180 FD C3AD          STD   OLDLOD
C183 BD D406  RDLP3   JSR   FMS       NOW READ BYTECT BYTES OF PROGRAM
C186 26 B7            BNE   ERROR
C188 7A C3AA          DEC   BYTECT
C18B 26 F6            BNE   RDLP3
C18D 20 95            BRA   RDLP1     WHEN DONE GET ANOTHER LOAD OR XFER
```

```
                        *
                        * ERROR HANDLER
                        *
                        * PROGRAM MUST EXIT VIA ERROR SINCE END OF FILE
                        * IS DETECTED BY PRESENCE OF ERROR CODE
                        *
C15E A6  01     ERROR   LDA   1,X       GET ERROR CODE
C160 81  00             CMPA  #0        IS IT END OF FILE?
C162 27  09             BEQ   EXIT      IF YES, PROGRAM COMPLETED SUCCESSFULLY
C164 BD  CD3F           JSR   RPTERR    ELSE REPORT ERROR AND RETURN TO FLEX
C167 BD  D403           JSR   FMSCLS    QUICK CLOSE OF ALL OPEN FILES
C16A 7E  CD03           JMP   WARMS     ERROR EXIT
                        *
                        * NORMAL EXIT WITH CLOSE OF OPEN FILES
                        *
C16D 8E  C26A   EXIT    LDX   #INFILE   POINT AT THE FCB
C170 86  04             LDA   #4        CODE FOR CLOSE FILE
C172 A7  30             STA   0,X
C174 BD  D406           JSR   FMS       NORMAL CLOSE OF INFILE
C177 26  E5             BNE   ERROR     YOU CAN HAVE AN ERROR CLOSING A FILE TOO
                        *
                        * NOW REPORT BYTES
                        *
C179 BD  CD21           JSR   PCRLF
C17C BD  CD24           JSR   PCRLF
C17F 8E  C1FB           LDX   #MSG1     FILE CONTAINS
C182 BD  C25E           JSR   PRINT
C185 8E  C301           LDX   #TOTBYT   TOTAL BYTES
C188 5F                 CLRB
C189 BD  CD39           JSR   OUTDEC    OUTPUT AS DECIMAL NUMBER
C18C 8E  C26A           LDX   #MSG2     BYTES (DECIMAL)
C18F BD  C25E           JSR   PRINT
C192 BD  CD24           JSR   PCRLF
C195 8E  C218           LDX   #MSG3     $
C198 BD  C25E           JSR   PRINT
C19B 8E  C301           LDX   #TOTBYT
C19E BD  CD45           JSR   OUTADR    OUTPUT AS HEX NUMBER
C1A1 8E  C228           LDX   #MSG4     BYTES (HEX)
C1A4 BD  C25E           JSR   PRINT
C1A7 BD  CD24           JSR   PCRLF
C1AA 8E  C235           LDX   #MSG5     TRANSFER ADDRESS IS $
>C1AD BD  C25E          JSR   PRINT
C1B0 BE  C3AF           LDX   #XFERAD   TRANSFER ADDRESS
C1B3 BD  CD45           JSR   OUTADR    OUTPUT AS HEX NUMBER
C1B6 BD  CD24           JSR   PCRLF
C1B9 7E  CD03           JMP   WARMS     ALL DONE, BACK TO FLEX
                        *
                        * NOT BINARY FILE ERROR
                        *
C1BC BD  CD24   NOTBIN  JSR   PCRLF
C1BF 8E  C248           LDX   #MSG6     NOT BINARY
>C1C2 BD  C25E          JSR   PRINT
C1C5 BD  CD24           JSR   PCRLF
C1C8 7E  CD03           JMP   WARMS
C1CB 46 49 4C 45 MSG1   FCC   /FILE CONTAINS /
C1CF 20 43 4F 4E
C1D3 54 41 49 4E
C1D7 53 20
C1D9 00                 FCB   0
C1DA 20 42 59 54 MSG2   FCC   / BYTES (DECIMAL)/
C1DE 45 53 20 28
C1E2 44 45 43 49
C1E6 4D 41 4C 29
C1EA 00                 FCB   0
C1EB 20 20 20 20 MSG3   FCC   /          $/
C1EF 20 20 20 20
C1F3 20 20 20 24
C1F7 00                 FCB   0
C1F8 20 42 59 54 MSG4   FCC   / BYTES (HEX)/
C1FC 45 53 20 28
C200 48 45 58 29
C204 00                 FCB   0
C205 54 52 41 4E MSG5   FCC   /TRANSFER ADDRESS IS $/
C209 53 46 45 52
C20D 20 41 44 44
C211 52 45 53 53
C215 20 49 53 20
C219 24
C21A 00                 FCB   0
C21B 46 49 4C 45 MSG6   FCC   /FILE IS NOT BINARY/
C21F 20 49 53 20
C223 4E 4F 54 20
C227 42 49 4E 41
C22B 52 59
C22D 00                 FCB   0
                        *
                        * PRINT SUBROUTINE
                        *
C22E A6  80     PRINT   LDA   ,X+
C230 81  00             CMPA  #0
C232 26  01             BNE   PRINT1
C234 39                 RTS
C235 BD  CD1B   PRINT1  JSR   PUTCHR
C238 20  F4             BRA   PRINT
                        *
                        * FILE CONTROL BLOCK AND VARIABLES
                        * IMMEDIATELY FOLLOWING PROGRAM CODE
                        *
C23A            INFILE  RMB   320       INPUT FILE CONTROL BLOCK
C37A 00         BYTECT  FCB   0         BYTE COUNT
C37B 0000       CURLOD  FDB   0         CURRENT LOAD ADDRESS
C37D 0000       OLDLOD  FDB   0         PREVIOUS LOAD ADDRESS
C37F 0000       XFERAD  FDB   0         TRANSFER ADDRESS
C381 0000       TOTBYT  FDB   0         TOTAL BYTES
C383 FF         FIRST   FCB   $FF       FLAG FOR FIRST READ
                        *
                        END   START

0 ERROR(S) DETECTED

SYMBOL TABLE:

  BYTECT C37A  DUPLOP C37B  ERROR  C15E  EXIT  C16D  FIRST C383
  FMS    D406  FMSCLS D403  GETFIL CD2D  INFILE C23A  MSG1  C1FB
  MSG2   C1DA  MSG3   C218  MSG4   C228  MSG5   C235  MSG6  C248
  NOTBIN C1BC  OLDLOD C37D  OUTADR CD45  OUTDEC CD39  PCRLF CD24
  PRINT  C25E  PRINT1 C265  PUTCHR CD1B  RDLOOP C12D  RDLP1 C123
  RDLP2  C13C  RDLP3  C182  RDLP4  C14A  RPTERR CD3F  SETEXT CD33
  START  C1BE  TOTBYT C381  WARMS  CD03  XFERAD C3AF
```

```
/* UTILITY TO COUNT BYTES IN A BINARY FILE */

ORIGIN = $C100;  /* FLEX UTILITY AREA */
STACK = $C7BB;  /* TOP OF UTILITY AREA FOR VARIABLES */

CONSTANT TRUE=-1, FALSE=0, BIN=0;

GLOBAL
   INTEGER
      BYTE_COUNT,
      LOAD_ADDRESS,
      TRANSFER_ADDRESS,
      OLD_LOAD_ADDRESS,
      TOTAL_BYTES;

   BYTE EOF, ERFLAG, INCHAR, FIRST_PASS;

BYTE ARRAY INFCB(320);  /* USE SYSTEM FCB AREA */

INCLUDE OUTSUBS.LIB;
INCLUDE KEYDUB.LIB;
INCLUDE FLEX.LIB;


PROCEDURE GETINT:INTEGER BYTE;
   BYTE = READ(,INFCB);
   BYTE = SHIFT(LOBYTE,8)+INTEGER(READ(,INFCB));
END PROC BYTE;

PROCEDURE MAIN;
   SET_FILENAME(,INFCB);
   SET_EXTENSION(,INFCB,BIN);
   OPEN FOR READ(,INFCB);
   SET_BINARY(,INFCB);
   TOTAL_BYTES =0;
   BYTE_COUNT = 0;
   OLD_LOAD_ADDRESS = $0000;  /* =227A# */
   FIRST_PASS = TRUE;
   EOF = FALSE;
   REPEAT
     INCHAR = READ(,INFCB);
     IF INCHAR = 016
     THEN
     BEGIN
        TRANSFER_ADDRESS = GETINT;
        EOF = TRUE;
END;
IF INCHAR = 02
THEN
BEGIN
   LOAD_ADDRESS = GETINT;
   BYTE_COUNT = INTEGER(READ(,INFCB));
   IF LOAD_ADDRESS >=OLD_LOAD_ADDRESS
   THEN
   BEGIN
      TOTAL_BYTES = TOTAL_BYTES + BYTE_COUNT;
      OLD_LOAD_ADDRESS = LOAD_ADDRESS + BYTE_COUNT;
   END;
      WHILE BYTE_COUNT > 0
      BEGIN
         READ(,INFCB);
         BYTE_COUNT = BYTE_COUNT - 1;
      END;
   END
   ELSE IF FIRST_PASS THEN
   BEGIN
      CRLF;
      PRINT("FILE IS NOT BINARY");
      JUMP 0CD03;
   END;
   FIRST_PASS = FALSE;
UNTIL EOF;
CLOSE_FILE(,INFCB);
CRLF; CRLF;
PRINT "FILE CONTAINS ";
PRINT(INT(TOTAL_BYTES));
PRINT" BYTES (DECIMAL)"; CRLF;
PRINT "          $";
PUT_HEX_ADDRESS(TOTAL_BYTES);
PRINT" BYTES (HEX)"; CRLF;
PRINT "TRANSFER ADDRESS IS $";
PUT_HEX_ADDRESS(TRANSFER_ADDRESS);
CRLF;
```

Basically OS-9

Ron Voigts
2024 Baldwin Court
Glendale Heights, IL 60139

MORE ABOUT C AND MEMORY

I was trying to decide what to write about this month. I have two good choices from last month's column. First, I promised to talk about the memory module header. Secondly, I had a few more things to say about C programs and how memory gets allocated. Being that I am back in school right now and my time is limited, I have decided to talk about the later topic and leave the memory module for next month. Sorry to anyone who was counting on the module header topic. I think this month's topic will be interesting, even if to those who don't program in C.

We should back track a little and understand how C is compiled with the Microware C Compiler. I haven't worked with any of the other available compilers, but I believe they work in a similar fashion. When you start the compiler with a line like:

```
ccl program.c
```

a number of things happen. First, c.com is created. This file will contain the instructions for the rest of the compilation. Next, c.prep is executed. It goes through the C program and processes the the pre-compiler commands. It takes care of things like #iocluded <stdio.h> and #define NULL 0. Next the actual C compiler goes into action and converts the code into an assembly language source. Level I users have a 2 pass compiler; Level II users, a 1 pass compiler. C.asm converts the code in a linkable module. Finally, c.link takes the module, cstart.r, modules from clib.l and any others you specify and links them into an executable code.

A lot can be learned from examining the assembly language code generated from the C source code. But before plodding ahead, it is best to talk a bit about the location counters that occur in the assembly language code. First, there is the PSECT. It is the instruction location counter. It indicates the start of source code. It contains 6 arguments. They are the module name, type/lang, attr/rev, edition and stacksize. The PSECT does nothing for reserving direct page or data area. For this, there is the VSECT. Finally, there is the CSECT which is the base offset counter. If you are at all interested take a look at /d1/SOURCES/os9defs.a on the C source disk that came with the Microware C Language disks.

Now that we have a little knowledge under our belt, let's take a look at a simple C program. This one defines three integers and assigns the value of 5 to "c".

```
main()
{
   int a, b, c;
   c=5;
}
```

To get an assembly language listing of this, we enter:

```
ccl ctest.c -a
```

The "-a" tells the C compiler to stop at the assembly language level. The assembly code it creates looks like:

```
psect ctest_c,0,0,0,0
 nam ctest_c
 ttl main
main:
 pshs u
 ldd #_1
 lbsr _stkcheck
 leas -6,s
 ldd #5
 std 0,s
 leas 6,s
 puls u,pc
_1 equ -70
 endsect
```

Notice the PSECT has the name "ctest_c" and the rest of the arguments are 0, which means this is a subroutine file. Our C program is entered through cstart.r, the linkable module that branches to our main. The first thing "main" does is to push the U register on the stack. Next D is loaded with the value -70, which is how much stack space is needed by main (ignore the minus sign). Three integers need 6 bytes, plus 64 more for the function. The total is 70. A long branch to subroutine _stkcheck is made to verify that there is enough space. A return means everything is O.K. The "leas -6,s" reserves 6 byte on the stack for the integers (that's 2 bytes per integer). For "c=5", register D is loaded with 5 and it is stored at "0,s", the spot for "c". "b" would have been 2,s and "a" would have been 4,s. The "leas 6,s" undoes the "leas -6,s" and the "puls u,pc" restores the U register and cause a return from the subroutine.

Last month I showed that there were no VSECT's in a previous program that some readers had a problem. The solution was to increase the data area and give more room for the stack to grow. Is it possible to get a VSECT from the C compiler? Let's try rewriting the previous program with one change. We'll put the integer declaration outside the "main" function.

```
int a, b, c;
main()
{
   c=5
}
```

Now C must do something about the three integers. They can't get stack space in main since they are outside of it. The assembly listing:

```
 psect ctest_c,0,0,0,0,0
 nam ctest_c
 vsect
a_: rmb 2
 endsect
 vsect
b_: rmb 2
```

```
 endsect
 vsect
c_: rmb 2
 endsect
 ttl main
main:
 pshs u
 ldd #_1
 lbsr _stkcheck
 ldd #5
 std c_,y
 puls u,pc
_1 equ -64
 endsect
```

Three VSECT's have occurred. They create three global labels. They are "a_, b_, and c_". The little underscore is added to each letter to help distinguish them from possible registers. At link time, space for the three integers will be reserved in the data area. Notice that 64 bytes are again reserved for the function at "_1". This time D is loaded with 5 and it is stored at "c_,y". The Y register points to the start of memory in C programs.

The variables in the first C example are known as automatic variables. They are unique to the function. Outside of the function, they are not available. When the function is exited, the variables are gone. In the second example, external variables are used. These variables are outside the function and they can be used by it and other functions. There are advantages to and disadvantages to both. Using 'auto' variables will save memory. The line:

```
    std 0,s
```

takes only 2 bytes of memory. While the line:

```
    std c_,y
```

takes 5 bytes. Usually, larger code means longer execution time. So, if you use 'auto' variables, they will accessed faster with less overhead. Using global variables does have its advantages, too. All functions can use the variables directly without having to pass them as arguments in function calls. The trade-off is slower execution and larger program size.

Fortunately, you can have it both ways. Microware C has a non-standard storage class called direct. If you declare a global to be direct, the variable will be placed in the direct page. 255 bytes are allowed for the direct page variables. If you use more, C.link will tell you about it. Another method to increase speed is to use register. The U register can be used once in a function. The following listing shows use of both, direct and register.

```
direct int a, b;
main()
{
   register int c;
   b=10;
   c=5;
}
```

```
 psect ctest_c,0,0,0,0,0
 nam ctest_c
 vsect dp
a_: rmb 2
 endsect
 veect dp
b_: rmb 2
 endsect
 ttl main
main:
 pshs u
 ldd #_1
 lbsr _stkcheck
 ldd #10
 std <b_
 ldu #5
 puls u,pc
_1 equ -64
 endsect
```

Notice the VSECT contains "dp", indicating a direct page variable. Variable "b_" is loaded with 10 using the direct page symbol "<". This line takes only 2 bytes of machine code. Register U is used for variable "c". One line of code is all that is needed to load the value 5 into it. The most effective use of register variables in loop counters. They will result in improved speed of execution.

There is a way to cause a VSECT to occur within a function. C provides a storage class called static. Static causes the compiler to provide data space for variables. The following is the C version using static and its assembly listing

```
main()
{
   static int a, b, c;
   c=5;
}
```

```
 psect ctest_c,0,0,0,0,0
 nam ctest_c
 ttl main
main:
 pshs u
 ldd #_1
 lbsr _stkcheck
 vsect
_2 rmb 2
 endsect
 vsect
_3 rmb 2
 endsect
 vsect
_4 rmb 2
 endsect
 ldd #5
 std _4,y
 puls u,pc
_1 equ -64
 endsect
```

Here the variable names in the assembly code are not the ones we used in the C source. We used "a, b, and c". They became "_2, _3 and _4", respectively. These are how labels are internally represented. Also, there are no ":" following the labels. All this means, the variables are known only to the function and not globally. Using static lets the C function create variables that are not destroyed on exit, but are known only to it.

USING C VARIABLES

I've included a C program that will multiply two matrices. A matrix is a set of numbers arranged in rows and columns. An example of a matrix is:

```
4   3   6
7   2   5
```

This one is 2x3. It contains 2 rows and 3 columns. A matrix can be multiplied by another. This next matrix is 3x1.

```
 3
11
 1
```

A criterion for multiplying a matrix by another is that the columns in the first must equal the rows in the second. The resulting matrix will be the number of rows in the first by the number of columns in the next. Multiplying these two examples will give a matrix that is 2x1. The cells in the answer are the sum of the products of rows in the first matrix and the columns in the second. So, the first cell in the answer is:

```
    4x3 + 3x11 + 6x1 = 51
```

The second cell is 48. I'll let you compute this one. The resulting matrix looks like:

This month's program will compute this all for you. It is limited to matrices that are 10x10 using integers.

The program is given 3 parts. One of the nice features of C is that programs can be written in parts and linked together as a final step. The advantage is that all purpose routines can be created and used over and over. A good example is the CLIB.L that contains all the C linkable modules like printf(), fopen(), and so on. Another advantage is that modules can be written, debugged and linked later. Once a module works properly there is no reason to continuously recompile. When all the modules are finished they can be combined at link time. Using such a method will shorten programming time, since needless compiling is eliminated.

Our program has 3 parts. They are matrix.c, iomult.c and mmult.c. They are in Listing's 1, 2 and 3. It is easiest to do the last 2 listings. Compiling them with

cci iomult.c -r and cci mmult.c -r.

Finally, when the last part is written enter:

cci matrix.c iomult.r mmult.r -f=matrix

Matrix.c will be compiled and linked with iomult.r and mmult.r. The "-f=" tells what the executable module is to be named matrix.

The modules use some of the variable techniques I talked about. In matrix.c declaring the modules ahead of "main()" causes them to be external variables. In mmult.r "extern" is used to tell the compiler that somewhere outside of the file, the variables have been declared. The iomult.c routines get the matrices passed as pointers rather than by external reference. These programs give a good example of using both 'auto' and 'extern' variables.

We've covered the major storage classes of C language. The best way to get an understanding of them is to take try them out. You might want to use the "-a" option of the C compiler and compile your listing down to assembly language. Or you can use this month's program.

Next time we'll get back to talking about the module header, as I promised. And I'll have another a program for you to try. Until then, have a good month!

\- - -

Listing 1

```
/* File name: matrix.c
   To compile: cci matrix.c iomult.r mmult.r -f=matrix
   Creates:    /d0/cmds/matrix
   Date:       15-JAN-86
   Use:        Microware C Compiler                */

#define SIZE 10

int l[SIZE][SIZE];
int m[SIZE][SIZE];
int a[SIZE][SIZE];
direct int lrow, lcol,
           mrow, mcol,
           arow, acol;

main()
{
   int i, j;
   printf("\n\n Matrix Multiplier    Version 1.0\n");
   printf("\n Maximum matrix size is 10x10\n");
   printf("\n Column size of Matrix 1 must equal\n")
   printf(" row size of Matrix 2\n");

/* Entering data for Matrix 1 */
   printf("\n\n Data For Matrix 1\n");
   printf("\n  Enter row size: ");
   scanf("%d", &lrow);
   printf(" Enter column size: ");
   scanf("%d", &lcol);
   printf("\n Enter data for Cells of Matrix 1\n\n");
   readmat(lrow, lcol, l);

/* Entering data for Matrix 2 */
   printf("\n\n Data For Matrix 2\n");
   printf("\n  Enter row size: ");
   scanf("%d", &mrow);
   printf(" Enter column size: ");
   scanf("%d", &mcol);
   printf("\n Enter data for Cells of Matrix 2\n\n");
   readmat(mrow, mcol, m);

/* Calculating Matrix multiplication */
   arow=lrow;
   acol=mcol;
   mmult();

/* Write results */
   printf("\n\n\n Multiplication Results\n\n");
   printf(" Matrix 1\n\n");
   writemat(lrow, lcol, l);
   printf("\n Matrix 2\n\n");
   writemat(mrow, mcol, m);
   printf("\n Result\n\n");
   writemat(arow, acol, a);

}
```

Listing 2

```
/* File name:  iomult.c
   To compile: cci iomult.c -r
   Module:     iomult.r
   Date:       13-JAN-86
   Compiler:   Microware C compiler            */

#define SIZE 10
#define TRUE 1
#define FALSE 0

/* Input/Output Routines for Matrix Multiply */
readmat(row, col, mat)
int row, col;
int mat[SIZE][SIZE];
{
   int i, j, k;

   for (i=0; i<row; i++)
      for (j=0; j<col; j++){
         printf("Value for Cell %3d %3d: ",i ,j);
         scanf("%d", &mat[i][j]);
      }

   while (changeit()==TRUE){
      printf("ENTER row   column   value  >>");
      scanf("%d %d %d", &i, &j, &k);
      mat[i][j]=k;
   }
}


/* find out if we change any cells */
changeit()
{
   char answer[10];
   int c;
   c=FALSE;
   printf("Change any values?  Y/N   \n");
   scanf("%s", answer);
   if ((answer[0]=='Y') || (answer[0]=='y'))
      c=TRUE;
   return(c);
}

/* Write a matrix to the terminal */
writemat(row, col, mat)
int row, col;
int mat[SIZE][SIZE];
{
   int i, j;
   for (i=0; i<row; i++){
```

```
52        for (j=0; j<col; j++ )
53           printf(" %6d",mat[i][j]);
54        printf("\n");
55     }
56 }
57
```

Listing 3

```
 1 /* File name:  mmult.c
 2    To compile: ccl mmult.c -r
 3    Creates :   mmult.r
 4    Date:       13-JAN-86
 5    Use:        Microware C Compiler                    */
 6
 7 #define SIZE 10 /* Matrix size */
 8
 9 mmult()
10 {
11    extern int l[SIZE][SIZE];
12    extern int m[SIZE][SIZE];
13    extern int a[SIZE][SIZE];
14    extern direct int lrow, lcol,
15                      mrow, mcol,
16                      arow, acol;
17    register int i; /* we can use the register once */
18    int j;
19
20 /* compute results and store in matirx 'a' */
21    for (i=0; i<arow; i++)
22       for (j=0; j<acol; j++)
23          a[i][j]=rcmult(i, j, lcol); /* lcol=mrow */
24
25 } /* end of mmult */
26
27 /* Calculate the result for one cell in the answer */
28 rcmult(i, j, k)
29 int i, j, k;
30 {
31    int temp;
32    register int q;
33
34    temp=0;
35    for (q=0; q<k; q++)
36       temp+=l[i][q]*m[q][j];
37 } /* end of rcmult */
38
39
```

OS9 NETWORK

OVERVIEW

OS-9/NET connects all types of devices through the networking line. Each device is accessed by the standard OS-9 pathlist format:

```
/<network name>/<node name>/<device path>
```

For example:

```
/net/system1/h0/doc/manual
/net/system2/d2/backup/file
```

"/net" specifies that the device is on a computer connected to the network. "system1" is the "node name" or station name. The node name may be assigned any legal device name (e.g., system45, Hugo, Howard, Heckle, Jeckle). Following this is a complete OS-9 pathlist.

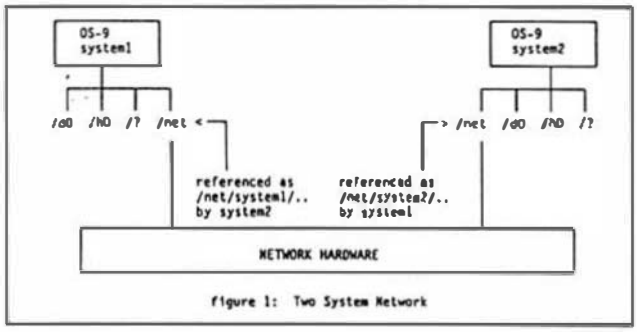


figure 1: Two System Network

Devices can currently be connected to each other using RS-232 cables as the network hardware. In this case, pathlists may be shortened to the following (this assumes "point-to-point" networking, as described in the device descriptor):

```
/<network name>/<device path>
```

Like other OS-9 devices, the name of the network itself is not "fixed". This allows a computer to be connected to more than one network. For example, you might name two types of network hardware connected to your system "/ETHER" and "/MODEM".

Because the Network File Manager and device drivers are re-entrant, you can add more of the same type of networking ports by adding corresponding device descriptors: "/ETHER2" and "/MODEM2".

In this manner, you can connect your system to four network lines.

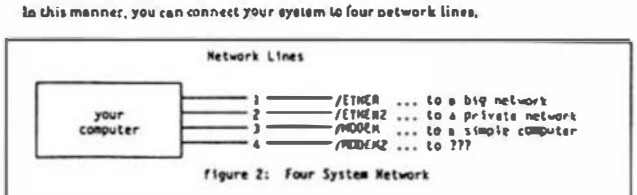


figure 2: Four System Network

NOTE: OS9/NET does not determine the type of network (ring, bus, star) or physical characteristics of them. This is the responsibility of the drivers and system dependent hardware.

REQUIREMENTS FOR NETWORKING

To implement OS9/NET, OS-9 Version 1.2 (or later) and at least one running OS-9 system is needed.

Like all standard SCF/RBF devices, OS9/NET requires both device drivers and device descriptors. In addition, if "multi-node" networking is being implemented, a "Node-ID Table" data module is required. This table describes the relationship between the physical station ID (node ID) and the corresponding station name (node name).

The Network File Manager and the dedicated system command program, "nmon", need no special configuration.

NETWORK FILE MANAGER

The internal functions of the Network File Manager are roughly divided into two parts. These to operate with each other to allow the system to communicate through the network:

1. The "Request Sender": This part sends I/O requests to the network. It is the standard interface between the file manager and the kernel/user. It receives a user I/O request, builds a message and sends it over the network. It then receives a service response from the node that received the message and reports the node's status and/or data to the user program. The Request Sender uses the device driver's "WRITE" entry point, regardless of type of I/O request.

   NOTE: This is only invoked when a user program executes an I/O system call.

2. The "Background Monitor/Server": This part acts as the network's incoming message monitor/server and response sender. It receives and parses incoming messages from other stations on the network. It then invokes a system process to execute the requested service on its associated system and returns the status/data to the requester over the network. If the incoming message is a response to a request, the message is passed to the Request Sender. The Monitor/Server uses the driver's "READ" entry point to allow the system to get responses (from requests as well) as recieve requests from other systems.

   NOTE: This is invoked by the program "Nmon".

These functions are divided into several smaller functions, which are discussed in the next section.

THE RELATIONSHIP BETWEEN THE ISO-OSI MODEL AND OS9/NET

Generally, the OS-9 kernel and the Network File Manager correspond to the OSI model's layer 7 (application), 6 (presentation) and 5 (session).

Tracing A Network User Request Through The ISO-OSI Model

1. The user program executes a service request to be given status and data (if available).

2. (Layer 7: Application Layer: OS-9 kernel)
   Upon receiving the service request, the kernel dispatches it to the correct I/O handling modules (in some cases RBF or SCF is called). For OS-9, networking is handled exactly like any other device handling. This layer does not care whether the request goes to the network or not. If it does, the Network File Manager is called. Consequently, the OS-9 network device is transparent to the system/user.

3. (Layer 6: Presentation Layer: Network File Manager)

A. The Network File Manager receives specialized I/O requests and builds a logical message in general OS9/NET format. It sends the message to the Session handler (layer 5).

B. The Network File Manager receives the message transfer status and data if it exists form layer 5 and returns them to layer 7.

4. (Layer 5: Session Layer: Network File Manager)

A. The Network File Manager recieves the logical message from layer 6. It then handles the physical device controls and calls the device driver (layer 4). It waits for the message to complete transmission and then receives the execution status/data from the server (passed back from layer 4). It returns the status /data to layer 6.

B. In the background, "Nmon" receives two types of incoming logical messages from layer 4:

I. Service responses from outlying stations are dispatched to waiting processes (from layer 6).

II. Service requests from outlying stations are invoked as a separate service process, which recursively calls the application layer. After the process' completion, a me age is sent back to the requester through layer 4.

NOTE: This is the Background Monitor/Server described previously in the File Manager description. All system requests originating outside the system are processed by the Background Monitor/Server.

5. (Layer 4 through 1: Transport to Physical Layer: Device Driver to Hardware Configuration)

A. Upon receiving a logical message from level 5, the device driver sends it out onto the network line according to its physical characteristics. This could range from RS-232C, HDLC or EtheeNet to large scale network communication lines.

Usually a specialized hardware level protocol is used for communication as well as the physical method for accomplishing it. Data enc yption and de-cryption are also done at this time.

B. Incoming logical messages from outer stations are sent to layer 5 without touching their content. Interrupt driven or hardware level monitoring for the incoming messages must be done after the system is brought up.

NOTE: These layers never look into the message contents; they merely pass them to the appropriate destination.

INTERFACE BETWEEN THE NETWORK FILE MANAGER AND THE OS9/NET DEVICE DRIVER

An OS9/NET device driver is like all other O .9 device drivers. It is called by and works with the kernel and the Network File Manager. It conforms to the standard OS-9 memory module format (module type code: Drivr). For full information on OS-9 device drivers, consult the "OS-9/68000 OPERATION SYSTEM TECHNICAL MANUAL".

The most important function of the OS9/NET driver is the handling of logical messages sent to or received from the Network File Manager. The driver uses the READ entry point to receive messages and transmits them using the WRITE entry point. All messages built by the Network File Manager conform to a general format. The following example is a message built when a I$Create system request is called. The first seven fields are standard fields used in all messages. The remaining fields are system call dependent (in this case, I$Create):

Length	Name	Description
Word	Msg_Dest	Destination Node ID
Word	Msg_Src	Source Node ID: Message sender ID
Word	Msg_Code	Meaning of Message: Create
Word	Msg_Path	Requester's Logical Path# on the Requester's System
Word	Msg_Pack	Requester's Node Packet#
Long	Msg_User	Requester't Group.User ID on the Requester's System
Word	Msg_Len	Message Length: Message begins at the next field
Word	RqCr_Mod	I$Create System Call Parameter
Word	RqCr_Atr	I$Create System Call Parameter
Word	RqCr_Siz	I$Create System Call Parameter
14 Bytes	RqCr_Dft	Requester's Login-Directory Information
Variable	RqCr_Pth	I$Create Parameter String, terminated by NUL

The driver examines only two of these fields:

Msg_Dest — Gives the driver the destination of the message. This is used in the Write entry.

Msg_Len — Gives the driver the length of the me age. It is also used in the Write entry.

All other fields are strictly used by the File Manager or a specific OS-9 device. This cluster of information is passed between OS-9 operating systems, not drivers. The driver is totally unaware of the contents of the message. It does not change or examine any part of the message other than the fields specified above. The driver merely sends the message of the length specified by Msg_Len (plus header) to the node specified by Msg_Dest. A hardware driver usually adds its individual header and trailer (i.e., a CRC check for physical error detection/correction):

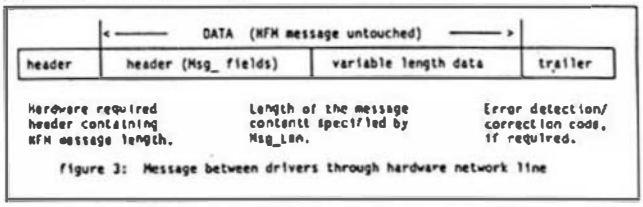


figure 3: Message between drivers through hardware network line

As has been shown, the Network File Manager has its own message format, the driver may have its own protocol and the hardware itself may also have its own protocol.

Because hardware characteristics vary from system to system, the Network File Manager does not concern itself about the independent hardware protocol. This is the responsibility of the device driver. The driver is solely responsible for handling the hardware, the network line, adding its individual header (and if required trailer) and if necessary exchanging some sequential hardware handshaking protocol.

This section describes the definitions of the initialization table contained in the device descriptor modules for the Network File Manager. These values are copied into the corresponding option field of the path descriptor at initialization (when Nmon is invoked or the device is "init"-ed). This table immediately follows the standard device descriptor module header fields (for full descriptions, see the "OS-9/68000 OPERATING SYSTEM TECHNICAL MANUAL"). The size of the table is defined in the M$Opt field. A graphic representation of the table is supplied in figure 4.

NOTE: The term "offset" refers to the location of a module field relative to the starting address of the module. Module offsets are resolved in assembly code by using the names shown here and linking the module with the relocatable library: "sys.l" or "usr.l".

Offset		Usage
$48	PD_DTP	Device Type
$49	PD_Hd yp	Network Hardware Type
$50	PD_Dest	Default Destination ID
$52	PD_Src	Node/Station ID
$54		Reserved
$56	PD_RtyO	# of Retries for Network File Manager
$58	PD_RtyF	# of Retries for Device Driver
$5A	PD_Mul	Multipler for Retries
$5C		Reserved

figure 4:. Device Descriptor Initialization Table

NAME	UTILIZATION
D_DTP	**Device Class** (0=SCF 1=RBF 2=PIPE 3=SBF 4=NET)
PD_HdTyp	**Network Hardware Type** If set equal to one (N_Multi_), this hardware is connected to a multi-station network line. The Network File Manager then expects a full network pathlist: /<network name>/<station name>/<full OS-9 pathlist> If set equal to zero, this hardware assumes only "point-to-point" networking. The Network File Manager then expects a simple network pathlist: /<network name>/full OS-9 pathlist>
PD_Dest	**Default Destination Station/Node ID** This field is used by the Network File Manager as the default destination station/node ID. It is copied to the path descriptor at initialization. When PD_HdTyp is set to one, the Network File Manager updates the corresponding field in the path descriptor for the destination of each network message, while this field (in the device descriptor) remains unchanged. This field is not used when point to point networking is specified.
PD_Src	**Source Station/Node ID** This is used by the Network File Manager as the source ID for all outgoing messages.
PD_RtyF	**Number of Re-Tries For Network File Manager** Some networking hardware return errors if a device is busy. When the Network File Manager detects an E$DevBsy error (passed by the driver), it will try to send the message over again. The field's value, when multiplied by the value of PD_Mul, specifies how many times it will try to send a message before returning an error.

PD_RtyD Number of Re-Tries Per Device Driver
Some networking hardware returns errors if a device is busy. When the device driver detects an E$DevBsy error driver), it will try to send the same message over again. The field's value, when multiplied by the value of PD_Mul, specifies how many times it will try to send a message before returning an error.

PD_Mul Re-Try Multiplier
See PD_RtyF and PD_RtyD.

NODE NAME DATA MODULE

The Network Node Name Data Module specifies the relationship between a node/station name and its corresponding node/station ID. This module is only used when the Network Device Descriptor field, PD_HdTyp is set to one, specifying multi-station networking. The name of this module must be the name of the network device with a "_nodes" suffix. For example:

For the device	Module Name
/net	net_nodes
/ether	ether_nodes
/omni	omni_nodes

This module is automatically loaded/linked at the time "Nmon is first invoked. If PD_HdTyp is set to one and this module is not in the current execution directory, the system will not start causing an "Nmon" error. If PD_HdTyp is set to zero, the system never uses this module, even when the hardware requires an actual destination ID (in this case, the driver is responsible to set the ID and establish linkage).

For a driver to be able to use the ID specified in Msg_Dest (and Pd_Dest), it must be described in this module. For each node/station to be used in the network, there is an entry for both the node ID and its name string. Each node ID entry has two fields:

Size	Use
Long	Offset to corresponding node name string
Word	Node ID

"C" User Notes

E. N. (Bud) Pass, Ph.D.
Computer Systems Consultants
1454 Latta Lane, N. W.
Conyers, GA 30207
404-483-1717/4570

INTRODUCTION

This chapter continues the discussion of the construction of a portable editor with the description of the curses and terminfo terminal-driver packages, which are readily available in many forms on UNIX and similar systems, on IBM PCs and clones, and on many other systems.

CURSES AND TERMINFO PACKAGES

The curees package provides a high-level method of updating screens with reasonable optimization and terminal independence. The name "curses" is intended to signify "cursor optimization", but many who have attempted to use it have come to believe that it is recursively and appropriately named, due to problems with some of its implementations.

In order to initialize the curses package, the routine initscr() must be called before any of the other curees routines. The routine endwin() must be called before exiting to terminate the use of the curses package.

Curses maintains an image of the current screen, and allows the user to establish an image of a new screen. Then the refresh() call makes the current screen look like the new screen, hopefully in the fastest possible manner for the target terminal device. The curses package is based on the terminfo package.

The terminfo package provides a lower-level method (relative to curses) of driving terminals and printers. The name "terminfo" is short for "terminal information", which is also the name of the accompanying and related terminal description file. It makes little attempt towards optimization, but it achieves some level of terminal independence thru the termcap and terminfo format files.

> **The curses package provides a high-level method of updating screens with reasonable optimization and terminal independence.**
>
> **The terminfo package provides a lower-level method of driving terminals and printers.**

In order to initialize the terminfo package, the routine setupterm() must be called before any of the other terminfo routines. The routine resetterm() must be called before exiting to terminate the use of the terminfo package.

Function Call	Description
addch(ch)	add a char to stdscr
addstr(str)	call addch for each char in str
attroff(at)	turn off video attributes on stdscr
attron(at)	turn on video attributes on stdscr
attrset(at)	set video attributes on stdscr
baudrate()	return speed of current terminal
beep()	sound bell
box(win,vert,hor)	draw box around window
cbreak()	set cbreak mode
clear()	clear stdscr

```
     clearok(scr,bf)                clear screen before
next refresh of scr
     clrtobot()                     clear to bottom on
stdscr
     clrtoeol()                     clear to end of line
on stdscr
     delay_output(ms)               insert ms msec delay
on output
     delch()                        delete a char
     deleteln()                     delete a line
     delwin(win)                    delete win
     doupdate()                     update the physical
screen
     echo()                         set echo mode
     endwin()                       end window mode
     erase()                        erase stdscr
     erasechar()                    erase char of current
terminal
     fixterm()                      set terminal to
terminfo mode
     flash()                        flash screen or sound
bell
     flushinp()                     flush typeahead on
current terminal
     getch()                        get a char through
stdscr
     getcap(name)                   get terminal
capability name
     getstr(str)                    get a string through
stdscr
     gettmode()                     establish current tty
modes
     getyx(win,y,x)                 get row and col
     has_ic()                       terminal can insert
char
     has_il()                       terminal can insert
line
     idlok(win,flag)                enable insert/delete
lines operations
     inch()                         char at current row
and col
     initscr()                      initialize curses
     insch(c)                       insert a char
     insertln()                     insert a line
     intrflush(win,bf)              interrupt flush output
     keypad(win,flag)               enable keypad input
     killchar()                     kill char of current
terminal
     leaveok(win,boolf)             set cursor leave
position flag
     longname(termbuf,name)         get long name from
termbuf
     meta(win,bf)                   control use of the
'meta' key
     move(y,x)                      move cursor to row and
col
     mvcur(lasty,lastx,newy,newx)   actually move cursor
     newpad(lines,cols)             initialize new pad
     newterm(type,fp)               initialize terminal
     newwin(lines,cols,begin_y,begin_x) initialize window
     nl()                           set newline mapping
     nocbreak()                     unset cbreak mode
     nodelay(win,bf)                control nodelay input
mode
     noecho()                       unset echo mode
     nonl()                         unset newline mapping
     noraw()                        unset raw mode
     overlay(win1,win2)             overlay win1 on win2
     overwrite(win1,win2)           overwrite win1 on top
of win2
     pnoutrefresh(pad,prow,pcol,srow,scol,xrow,xcol) pre-
prefresh(pad)
     prefresh(pad,prow,pcol,srow,scol,xrow,xcol)
refresh(pad)
     printw(fmt,arg1,arg2,...)      printf on stdscr
     putp(str)                      tputs(str,1,putchar)
for terminfo
     raw()                          set raw mode
     refresh()                      update current screen
     resetterm()                    reset terminal from
terminfo mode
     resetty()                      reset tty flags to
stored value
     savetty()                      store current tty
flags
     saveterm()                     save current state of
tty
     scanw(fmt,arg1,arg2,...)       scanf through stdscr
     scroll(win)                    scroll win one line
     scrollok(win,bf)               set scroll flag
     set_term(new)                  set current terminal
     setscrreg(top,bottom)          set up scrolling
region on stdscr
     setterm(name)                  set term variables for
name
     setupterm(term,fd,errret)      initialize terminal for

terminfo
     standend()                     end standout mode
     standout()                     start standout mode
     subwin(win,lines,cols,begin_y,begin_x) create
subwindow win
     touchwin(win)                  mark all of win
changed for refresh()
     tparm(string,p1,..,p9)         expand a parameterized
string for terminfo
     tputs(string,affcnt,outc)      process a capability
string for terminfo
     traceoff()                     turn off debugging
output
     traceon()                      turn on debugging
output
     typeahead(fd)                  start typeahead
     unctrl(ch)                     printable version of
ch
     vidattr(newmode)
vidputs(newmode,putchar) for terminfo
     vidputs(newmode,outc)          output video
attributes thru outc
     waddch(win,ch)                 add char to win
     waddstr(win,str)               call waddch for each
char in str
     wattroff(win,at)               turn off video
attributes on win
     wattron(win,at)                turn on video
attributes on win
     wattrset(win,at)               set video attributes
on win
     wclear(win)                    clear win
     wclrtobot(win)                 clear to bottom of win
     wclrtoeol(win)                 clear to end of line
on win
     wdelch(win,c)                  delete char from win
     wdeleteln(win)                 delete line from win
     werase(win)                    erase win
     wgetch(win)                    get a char through
win
     wgetstr(win,str)               get a string through
win
     winch(win)                     get char at current
row and col in win
     winsch(win,c)                  insert char into win
     winsertln(win)                 insert line into win
     wmove(win,y,x)                 move cursor to row and
col on win
     wnoutrefresh(win)              copy win to virtual
screen
     wprintw(win,fmt,arg1,arg2,...) printf on win
     wrefresh(win)                  make screen look like
win
     wscanw(win,fmt,arg1,arg2,...) scanf through win
     wsetscrreg(win,top,bottom)     set up scrolling
region on win
     wstandend(win)                 end standout mode on


win
     wstandout(win)                 start standout mode
on win
```

DEMO PROGRAMS FOR CURSES AND TERMINFO

Following are two programs which differ only in that the first uses the curses package and the second uses the terminfo package. The program reads a sample data file (normally representing a screen image) and displays it. The following special sequences are supported:

```
\B   bold
\U   underline
\R   reverse
\X   reverse dim
\D   dim
\H   high
\N   normal
```

Although these programs are relatively simple and use only a small amount of the capability of either package, they do provide some of the flavor of each package. The listing of a sample input data file follows the listings of the programs.

Although these programs are relatively simple and use only a small amount of the capability of either package, they do provide some of the flavor of each package.

```
/* abridged version of curses.h include file used below */

struct _win_st
{
    short _cury, _curx;
    short _maxy, _maxx;
    short _begy, _begx;
    short _flags;
    chtype _attrs;
    bool _clear;
    bool _leave;
    bool _scroll;
    bool _use_idl;
    bool _use_keypad; /* 0=no, 1=yes, 2=yes/timeout */
    bool _use_meta; /* T=use the meta key */
    bool _nodelay; /* T=don't wait for tty input */
    chtype **_y;
    short *_firstch;
    short *_lastch;
    short _tmarg, _bmarg;
};

typedef struct _win_st WINDOW;
extern WINDOW *stdscr, *curscr;

extern int LINES, COLS;

#define A_NORMAL     0000000 /* normal */
#define A_CHARTEXT   0000177 /* text */
#define A_STANDOUT   0000200 /* high */
#define A_UNDERLINE  0000400 /* underline */
#define A_REVERSE    0001000 /* reverse */
#define A_BLINK      0002000 /* blink */
#define A_DIM        0004000 /* dim */
#define A_BOLD       0010000 /* bold */
#define A_INVIS      0020000 /* invisible */
#define A_PROTECT    0040000 /* protect */
#define A_ALTCHARSET 0100000 /* alt char set */
#define A_ATTRIBUTES 0377600 /* attributes */

#define KEY_BREAK     0401  /* break */
#define KEY_DOWN      0402  /* The four arrows */
#define KEY_UP        0403
#define KEY_LEFT      0404
#define KEY_RIGHT     0405
#define KEY_HOME      0406  /* Home up */
#define KEY_BACKSPACE 0407  /* Backspace */
#define KEY_F0        0410  /* Function key 0 */
#define KEY_F(n)      (KEY_F0+(n)) /* Function keys 1-63 */
#define KEY_DL        0510  /* Delete line */
#define KEY_IL        0511  /* Insert line */
#define KEY_DC        0512  /* Delete character */
#define KEY_IC        0513  /* Insert char or enter
insert mode */
#define KEY_EIC       0514  /* Exit ineert char mode */
#define KEY_CLEAR     0515  /* Clear screen */
#define KEY_EOS       0516  /* Clear to end of screen */
#define KEY_EOL       0517  /* Clear to end of line */
#define KEY_SF        0520  /* Scroll 1 line forward */
#define KEY_SR        0521  /* Scroll 1 line backward */
#define KEY_NPAGE     0522  /* Next page */
#define KEY_PPAGE     0523  /* Previous page */
#define KEY_STAB      0524  /* Set tab */
#define KEY_CTAB      0525  /* Clear tab */
#define KEY_CATAB     0526  /* Clear all tabs */
#define KEY_ENTER     0527  /* Enter or send  */
#define KEY_SRESET    0530  /* Soft (partial) reset */
#define KEY_RESET     0531  /* Reset or hard reset  */
#define KEY_PRINT     0532  /* Print or copy */
#define KEY_LL        0533  /* Home down  */
#define KEY_A1        0534  /* Upper left of keypad */
#define KEY_A3        0535  /* Upper right of keyped */
#define KEY_B2        0536  /* Center of keypad */
#define KEY_C1        0537  /* Lower left of keyped */
#define KEY_C3        0540  /* Lower right of keypad */

/* end curses.h */

/* curses version of highlight example program */

#include <curses.h>

#define setettr(x) wettrset(curecr,(x) & A_ATTRIBUTES)

main()
{
        initscr();
        high(stdscr);
        endwin();
        exit(0);
}

high(curecr)
WINDOW *curscr;
{
        short int c,c2;

        scrollok(curscr,TRUE);
        for (;;)
        {
                if ((c = getchar()) == EOF)
                        break;
                if (c == '\\')
                {
                        switch(toupper(c2 = getc(stdin)))
                        {
                        case 'B':
                                setattr(A_BOLD);
                                continue;
                        case 'U':
                                setettr(A_UNDERLINE);
                                continue;
                        case 'R':
                                setattr(A_REVERSE);
                                continue;
                        case 'X':
                                setattr(A_REVERSE |
A_DIM);
                                continue;
                        case 'D':
                                setattr(A_DIM);
                                continue;
                        case 'P':
```

```
                        setattr(A_BLINK);
                        continue;
                case 'H':
                        setattr(A_STANDOUT);
                        continue;
                case 'N':
                        setattr(0);
                        continue;
                }
                addch(c);
                addch(c2);
        }
        else
                addch(c);
    }
    wrefresh(curscr);
}

/* end curses version of highlight example program */

/* terminfo version of highlight example program */

#include <curses.h>
#include <term.h>

main()
{
    setupterm(0, 1, 0);
    high();
    resetterm();
    exit(0);
}

outc(c)
char c;
{
     putchar(c);
}

high()
{
    short int c,c2;

    for (;;)
    {
        if ((c = getchar()) == EOF)
            break;
        if (c == '\\')
        {
            switch(toupper(c2 = getc(stdin)))
            {
            case 'B':
                tputs(enter_bold_mode, 1,
outc);
                continue;
            case 'U':

tputs(enter_underline_mode, 1, outc);
                continue;
            case 'R':
                tputs(enter_reverse_mode,
1, outc);
                continue;
            case 'X':
                tputs(enter_reverse_mode,
1, outc);
            case 'D':
                tputs(enter_dim_mode, 1,
outc);
                continue;
            case 'F':
                tputs(enter_blink_mode,
1, outc);
                continue;
tputs(enter_standout_mode, 1, outc);
                continue;
            case 'N':

tputs(exit_attribute_mode, 1, outc);
                continue;
            }
            putchar(c);
            putchar(c2);
        }
        else
            putchar(c);
    }
}

/* end terminfo version of highlight example program */

/* sample input file for both highlight programs */

 \N this is normal \N

 \R this is reversed \N

 \U this is underlined \N

 \H this is standout \N

 \F this is blinking \N

 \D this is dim \N

 \X this is reversed dim \N

 \B this is bold \N

/* end sample input file for both highlight programs */
```

C PROBLEM

The program below computes the Fibonacci sequence to any given level of precision. The digits of a given element of the sequence are stored backward as separate characters. Arithmetic is performed sequentially on the digits, with high-order carry determining how many digits are required to contain the element. The two accumulators are swapped at each iteration to simplify the calculation of the next element of the sequence.

```
/* fiboneci.c - compute Fibonacci sequence to given
precision */

#include <stdio.h>
#include <ctype.h>

#define MAXDIGIT 1024 /* may be made as large as
necessary */
main(argc, argv)
int argc;
char **argv;
{
    int c, i, j, k, md, digits;
    char dig1[MAXDIGIT + 1], dig2[MAXDIGIT + 1];

    if ((argc < 2) || (!isdigit(*argv[1])))
    {
nogo:
        printf("Usage: %s nn\n", argv[0]);
        printf("   where nn is digits of
precision;\n");
        printf("          0 < nn < %d\n",
MAXDIGIT);
        exit(1);
    }
    digits = 0;
    sscanf(argv[1], "%d", &digits);
    if ((digits < 1) || (digits >= MAXDIGIT))
        goto nogo;
    for (i = md = 0; i <= MAXDIGIT; ++i)
        dig1[i] = dig2[i] = 0;
    dig2[0] = 1;
    printf("Fibonacci sequence to %d digits\n",
digits);
    printf("  1 0\n");
    printf("  2 1\n");
    for (k = 2; md < digits; )
```

```
        {
                for (i = c = 0; i <= md; ++i)
                {
                        j = dig2[i];
                        dig2[i] += dig1[i] + c;
                        dig1[i] = j;
                        for (c = 0; dig2[i] > 9; ++c)
                                dig2[i] -= 10;
                }
                if (c)
                        dig2[++md] = c;
                printf("%3d ", ++k);
                for (i = md; i >= 0; --i)
                        putchar(dig2[i] + '0');
                putchar('\n');
        }
}
```

For the next problem, code functions which will perform integer multiplication, division, addition, and subtraction in extended precision. Each number should be stored in a structure which stores the characters representing the number and also provides its current and maximum number of significant digits. It is not necessary to store the digits one per character, although that is acceptable in this problem. Each function should have three parameters (two operands and a result) and return the status of the result (overflow or normal).

EXAMPLE C PROGRAM

Following is this month's example C program; it is really a mini-course on linked structures, as are used in the portable editor currently being developed and described here. Structures and pointers are the most difficult concepts of the c language for most programmers to grasp.

```
/*
    Demonstration of simple linked list using pointers and recursion

    The C language encourages programmers to use pointers in more
    ways than virtually every other high-level language. If you're
    going to program in C, you should be familiar with the use of
    pointers that is most common in just about all the other
    languages that allow pointers -- the linked list (and its more
    sophisticated relative, the binary tree). Here, for starters,
    is a very simple demonstration of a very simple, single linked
    list.

    by David M. Walker 71076,411
       modified by J.M. Mulcino
*/

#define NULL 0
#define MAXLINE 80

struct entry
{                           /* Each item in the list contains  */
    char text[MAXLINE];     /* some data (here, a text string) */
    struct entry *next;     /* and a pointer to the next item  */
}
dummy;                      /* dummy used for sizeof */

main()
{
    struct entry *root;     /* will point to first item in list */

    root = makelist();      /* get the data, reserve memory for */
                            /* each item, store the text, and   */
                            /* return a pointer to the first    */
                            /* item in the list                 */
    puts ("\n\n");
    putlist (root);         /* scan the list, displaying each   */
}                           /* item, until there are no more    */

makelist()
{
    struct entry *r;
    char tmptxt[MAXLINE];   /* string input buffer and pointer  */
    char *inptr;
    int i;                  /* counter, just for display        */

    r = i = NULL;           /* NULL means the pointer does not  */
                            /* point to valid data              */
   do
   {
                            /* get string as sample data        */
       printf ("\nEnter text for item %d: ", ++i);
       if (inptr = gets (tmptxt,MAXLINE))
           stotext(inptr, &r); /* Note the '&'   */
   }
   while (*inptr);          /* quit when empty string input     */
   return r;                /* return value for root in main()  */
}

/*
    This is the tricky part.  We want to scan the list until
    we find a NULL pointer marking the end of the list.
    (When the list has not been created yet, the initial pointer
    will be NULL.)  Then we find space for a new item, replace
    the NULL with a pointer to the new item, store the current
    data in the new item, and NULL the new item's "next" pointer,
    to show that the new item is now the end of the list.
    Because the pointer parameter is to be changed by the
    function, we must pass to it a pointer to a pointer.
*/

stotext(s,p)
char *s;
struct entry **p;           /* pointer to pointer               */
/*
    Pause here for a brief lecture on pointers to pointers.
    The basic data item that we are interested in here is a
    structure of the type "entry" as defined above.  We will
    access each item by a pointer (like the pointer "root"
    in the main function) because (a) that's the "link" in
    our linked list and (b) we're allocating memory for each
    item dynamically, so we need a pointer to tell us where
    the allocated memory is.  But, if we pass the pointer
    itself as a parameter, we can't change its value for the
    calling function (since the value of a parameter is always
    local to a function in C).  Therefore, we pass to the
    function a pointer to the pointer that we want to change,
    and change it by indirection through the passed pointer.

    Once more, in brief:  "struct entry **p" means that p
    is a pointer to a pointer to a data structure of the type
    defined as "entry"; *p is the pointer to which p points;
    and **p is the structure to which *p points.
    Since *p is a pointer to a structure, (*p)->text represents
    the "text" field of the structure to which *p points.
    The same field could also be referred to as (**p).text.
    The form *p->text would mean the data object pointed to by
    the pointer p->text, which would be an error here, since p
    points to a pointer, not to a structure.  This is not a
    particularly easy concept to grasp, and deserves some study.
*/
{
    if (*p == NULL)
    {                       /* NULL means end of list           */
        *p = alloc (sizeof (dummy));
                            /* make space for new item          */
                            /* and change *p (the pointer to    */
                            /* which p points) to point to the  */
                            /* new space                        */

        printf ("Storing item at %04x\n", *p);

        strcpy ((*p)->text, s);
                            /* store data in new item           */
        (*p)->next = NULL; /* mark new item as end of list     */
    }
    else
    {                       /* Not end of list yet              */

        printf ("Checking at %04x\n", *p);

        stotext (s, &(*p)->next);
                            /* try the next item                */
    }                       /* again, note the '&' -- we're     */
}                           /* passing a pointer to the 'next' */
                            /* pointer of item (*p)             */
/*
    In contrast with the complications of stotext(),
    see how simple it is to scan the completed list.
    We simply follow the "root" pointer to the first
    item, display its data, then follow its "next"
    pointer to the next item, recursively, until the
    "next" pointer is NULL, at which point we stop.
*/

putlist(r)
struct entry *r;
{
    if (r)
    {                       /* that is, if r is not NULL        */
                            /* then r points to a valid item,   */
                            /* so display data                  */
        printf ("Item at %04x - %s\n", r, r->text);
        putlist (r->next);
                            /* ... and recurse ... that is,     */
    }                       /* call this same function with     */
                            /* the "next" pointer of the cur-   */
                            /* rent item giving the value of r  */
                            /* by implication, if r is NULL,    */
                            /* do nothing more, just return     */
}
```

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

Interactive Compact Disks

The news is out. Interactive compact disks have been standardized by Sony and Philips, and Microware has written the software for them. Not Digital Research; not Microsoft; Microware!

My understanding of this new technology is limited to what I learned from Ken Kaplan in a long conversation. I don't have the official press release yet. Please forgive me if I get some details wrong. This is too exciting to wait for hardware or even full documentation to arrive.

How do these disks differ from the CDs that many of us already own? They are compatible enough that you will be able to play your existing CDs on the new players. Some Interactive CDs may play on old-style players, but the new features won't be available. The new features amount to storage of encoded pictures and data.

The structure of an Interactive Compact Disk (I'm going to start calling them CDIs) is somewhat like other OS-9 disks. There are directories and files. The file descriptors must be pretty elaborate. A given file can contain several flavors of audio, video, and other data mixed on a block-by-block basis. When OS-9 reads a file it routes the video to one of two special video-processor chips, the audio to a sound chip, and the other data back to the program.

There are new formats for digitally encoded music. They basically record the derivative of the signal instead of its value. The upshot is that the CDIs are able to squeeze music into fewer bytes than CDs. They can read music data fast enough to reproduce sound with quality equivalent to CDs and read a full frame of high-quality video information about once per second.

The easy thing to do with this is run a slide show with your music, or do a classy talking reference collection. Imagine what National Geographic might be able to do with this! Things get more interesting if you will compromise a little. Use lower quality sound (there are several levels) and you get more bandwidth for other stuff. Use a picture that only fills half the screen and you can refresh it twice a second. Use a picture that is mostly stationary (say something moving against a stationary background) and you can update faster than you would want to.

An CDI holds about six-hundred megabytes. This is read-only memory and seek time on CDIs (like CDs) is terrible. Moving around on a disk isn't what hard-disk users are used to. The data transfer rate isn't up to hard disk standards either. Evidently this is aimed at the mass market. I imagine that compatibility with CDs and expense were important considerations.

! Not Digital Research !
! Not Microsoft !
! Yes - Microware - Did it !

The news is out. Interactive compact disks have been standardized by Sony and Phillips, and Microware has written the software for them.

Microware did the best they could with these slow devices. They have software that optimizes the positioning of information on the disk. Opening a file can be done in one seek. Consider the problem of reading potentially large, complicated files without seeking to the file descriptor information in the middle of reading the file. OS-9 does this with conventional files, but a substantial gap in the middle of a tune while OS-9 found the descriptor for the rest of the file would NOT be a good solution. Do you suppose that they decided that memory is inexpensive enough that they could read and store the entire file descriptor when a file is opened? Maybe they interleave it with the file. I can imagine a system of block prefixes being used as file descriptors. Remember the file system is static.

The CDI standard is oriented around music, but it will make a dandy storage medium for text, databases, or vast collections of programs. The only thing I don't see any way for it to be is a replacement for video tapes/disks. It doesn't have enough video bandwidth for that.

No, you can't buy one yet. The standard has been announced and Sony and Philips will (now? soon?) sell support chips, but CDIs aren't about to hit the market.

Microsoft, Atari, and various others have been talking about CDs. There is no reason why OS-9 CDI software couldn't be able to read disks made in Microsoft's format, but only the OS-9 standard includes audio and video so with any luck others will quietly fade away.

OS-9 is the operating system for the standard. We can look forward to being better known at least in the consumer electronics field.

As I commented in my last column, having OS-9 sold into mass markets is a mixed blessing for those of us not in that mass. In this case I hope the market will be large enough that its spin-offs will be very good for us.

A Challenge

People seem to want to write programs that are able to do everything. A program for sequentially searching a file accepts regular expressions as search arguments. A sorting program can sort on arbitrary-precision floating-point numbers. A telecommunication program supports an elaborate programming language to permit shortcuts in communication. All these features are nice. In fact, these particular features stick in my mind because I've used them. However, they aren't used often, and they cost quite a lot. Elaborate programs may be slower than simpler counterparts, they are likely to cost more to purchase, and they definitely use more memory.

If your computer is fast enough and has enough memory comprehensive programs are an advantage. The more powerful each program is the fewer programs a person needs to use. Popular integrated packages like Symphony and Framework are intended to let your do all your work with just one program. For several years I maintained a program called SuperWylbur, an editor that attempted to be everything-but-the-kitchen-sink. Many people did all their computer work without seeing anything but SuperWylbur. Some people were so well insulated by SuperWylbur macros that they didn't even see SuperWylbur. I was (still am) proud of that program, but I have to admit that it's a long stretch to call it anything like an efficient way to do anything but text editing.

I just bought a CMX micro-20 (MUSTANG-020 SBC). It will take me a while to fill the two megabytes of memory that are on that board, but I'm sure I will. My 6809 machine has almost a half megabyte of memory and I'm short of space there. I don't keep many utility commands resident because the good ones, like grep and sort, use a lot of memory. If they were smaller I would keep them loaded. If they were loaded I would be inclined to use them more often. I look at unsorted directory listings and LIST through files looking for key words because I'm too impatient to wait for non-memory-resident programs to load.

I have included two tiny utilities with this column. They are designed for small size at almost all costs. There are no features — the programs only do one thing each. The advantage is that even a CoCo user might be willing to leave them memory resident.

My challenge is directed to Don Williams and to my readers. To my readers: see if you can add more utilities to this package or improve these (without adding many bytes of code). To Don: try to think of a way to run a tiny programs contest. Collect the best of them and package them. Perhaps Southeast Media could sell them. One or two tiny programs aren't worth much, but ten or twenty could be to an OS-9 system what Sidekick is to a PC.

To give you perspective, I think 256 bytes is big for a tiny program. I'd like to be able to fit a set of programs that do the important Unix-tool functions in one K. The three programs that I'd add to this set first would be Uniq (if sequential lines in a file are identical, write only one of them), Sort, and Dc (a simple RPN calculator). They should each fit within 256-bytes.

Editor's Note: This CDI thing is probably bigger than most of you might believe. It is an industry standard that will influence most all major manufacturers.

Originally the big gun was to be a system call CD-ROM. Apple, DEC, TMS, Microsoft, 3M, Laser-data, RCA, GE, Videotool, Xebec, Yelick and others were all going in that direction (and may yet). Then along came Sony and Phillips N.V. (68 Micro Journal subscribers - customers also) and they opted for their new system CD-I. At that time they were the leaders in CD-ROM technology. As of this date no word has been released as to what the other biggies mentioned will do. However, it seems that Sony-Phillips N.V.-OS-9 has taken the lead. So it appears this may well be the future of compact disk technology.

The CD disk market is a large slice of consumer products. While not related to our usage of OS-9, it still is great to know that it IS Microware's OS-9 that got the nod. There were several very popular systems considered, but Microware was the winner!

The specs call for a Motorola 68000 processor, custom graphics and sound processors (still in development) and, of course, OS-9. The original CD-ROM specs were to work with all popular pc systems, but it seems that this might restrict it to OS-9 alone.

Just thought you might like to know.

DMW

\- - -

```
Microware OS-9 Assembler 2.1   03/14/86 12:35:43
ttr - tiny translation program

                              nam   ttr
                              ttl   Tiny translation program
                              #pl
                              endc
                        *-------------------------------------------------*
                        *    Copy a file from standard input to standard*
                        *    output translating a specified character   *
                        *    to another specified character (or nothing)*
                        *  ttr <c1> <c2>                                *
                        *    c1 is the character to search for.         *
                        *    c2 is the character to translate it to.    *
                        *  A backslash can be used to escape for special*
                        *  characters.                                  *
                        *    \\   backslash                             *
                        *    \t   tab                                   *
                        *    \c   where c is not a space:  c            *
                        *    \b   backspace                             *
                        *    \n   <CR>                                  *
                        *-------------------------------------------------*
0001 0011               TYPE    set   PRGRM+OBJCT
0001 0001               REVS    set   REENT+1
0000 97C08898                   mod   ttrsize,ttrname,TYPE,REVS,Entry,MemSize
000D 7474F2             ttrname fcs   /ttr/
0001 0001               c1      rmb   1         from character
0001 0001               c2      rmb   1         to character
0001 0002               inchr   rmb   1         an input character
0018                    Entry
0018 A584                       lda   ,X
001A 8120                       cmpa  #$20
001C 2604                       bne   Go
001E 3001                       leax  1,X
0020 20F6                       bra   Entry
0022 9D40               Go      bsr   chkbsl    get and fix a character
0024 9788                       sta   c1
0026                    Skipbl
0026 3001                       leax  1,X
0028 A684                       lda   ,X
002A 8120                       cmpa  #$20
002C 27F8                       beq   Skipbl
002E
002E 8100                       cmpa  #$0D      <CR>?
0030 2603                       bne   Go2       no: there is a translation cha
0032 0F01                       clr   c2        yes: translate to nothing
0034 2004                       bra   RunLoop
0036                    Go2
0036 9D2C                       bsr   chkbsl    get translation character
0038 9788                       sta   c2
003A
003A                    RunLoop
003A 3042                       leax  inchr,U
003C 108E0001                   ldy   #1
0040                    RunLol
0040 8600                       lda   #0        (std in)
0042 103F89             ReadIt  OS9   I$Read    read a char to translate
0045 2515                       bcs   Runf
0047 9682                       lda   inchr     translate the char
0049 8100                       cmpa  c1
```

```
0005F  0043 26da            bne   RunLp2
00061  0045 9681            lda   c2
00063  0047 27EF            beq   RunLp1
00065  0049 9702            sta   inchr
00067            RunLp2
00069  004B B681            lda   #1          (std out)
0006B  004D 103F8C          OS9   I$WritLn
00070  0050 2502            bcs   Runt
00072  0052 20E4            bra   RunLp1
00074            Runt
00076  0054 C1D3            cmpb  #211        just EOF?
00078  0056 2681            bne   Error
0007A  0058 5F              clrb
0007B  0059              Error
0007D  0059 103F06          OS9   F$Exit
00080
00083            chkbsl
00085  005C A684            lda   ,X          first byte of parameter area
00087  005E 815C            cmpa  #'\         is it a back slash?
00089  0060 2701            beq   chkbs6
0008B  0062 39              rts
0008C  0063              chkbs6
0008E  0063 A601            lda   1,X         get the next character
00090  0065 815C            cmpa  #'\
00092  0067 2603            bne   chkbs5      \\ is \
00094  0069              bslfix
00096  0069 3001            leax  1,X
00098  006B 39              rts
00099  006C             chkbs5
0009B  006C 8120            cmpa  #$20
0009D  006E 2FF9            ble   bslfix
0009F  0070 8A20            ora   #$20[00000000] make it lowercase
000A1  0072 8174            cmpa  #'t         tab?
000A3  0074 2604            bne   chkbs2
000A5  0076 8609            lda   #$09        load a tab
000A7  0078 20EF            bra   bslfix
000A9  007A              chkbs2
000AB  007A 8162            cmpa  #'b         backspace?
000AD  007C 2604            bne   chkbs3
000AF  007E 8608            lda   #$08        backspace
000B1  0080 20E7            bra   bslfix
000B3  0082              chkbs3
000B5  0082 816E            cmpa  #'n         (CR)?
000B7  0084 2604            bne   chkbs4
000B9  0086 860D            lda   #$0D        (CR)
000BB  0088 20DF            bra   bslfix
000BD  008A              chkbs4
000BF  008A 865C            lda   #'\         just use \
000C1  008C 39              rts
00120 D 0003             MemSize equ   .
00121  008D 4C6C0C            emod
00124  0090              ttrsize equ   *

00000 error(s)
00000 warning(s)
00090 00144 program bytes generated
00003 00003 data bytes allocated
02205 00917 bytes used for symbols
```

Microware OS-9 Assembler 2.1 05/14/86 12:48:04
srch - tiny sequential search program

```
00001                       nam   srch
00002                       ttl   tiny sequential search program
00003                       ifp1
00004                       endc
00005       *-------------------------------------------------*
00006       *    srch:  A simple sequential search program    *
00007       *    Syntax:                                      *
00008       *        srch <string>                            *
00009       *    Input is strictly from standard input.       *
00010       *    Output is strictly to standard output.       *
00011       *    The search string stretches from the first   *
00012       *    non-blank after the command name (srch),     *
00013       *    to the end of the line (or the < for         *
00014       *    input redirection.  For example              *
00015       *      srch Name <addr.file                       *
00016       *    Will search for the string "Name "           *
00017       *    Be sure to squash the < right up against     *
00018       *    the search string,  srch Name<addr.file      *
00019       *    If you want leading blanks in your           *
00020       *    search string use an initial quote:          *
00021       *     srch 'Name <addr.file                       *
00022       *    Will search for ' Name ".  If you want       *
00023       *    an initial ', use two.                       *
00024       *-------------------------------------------------*
00025 0011          TYPE    set   PRGRM+OBJCT
00026 0081          REVS    set   REENT+1
00027 0000 87CD06AC         ood   srcbsiz,srchna,TYPE,REVS.Entry,MemSize
00029 0000 737263E8 srchna  fcs   /srch/
00030
00031 0100          MAXSIZ  set   256
00032 0100          STACK   set   256
00033 D 0000        Key     rmb   2           points to search string
00034 D 0002        Work    rmb   2           points to search loc
00035 D 0004        Line    rmb   MAXSIZ
00036 D 0104                rmb   STACK
00037 D 0204        MemSize equ   .
00038 0011          Entry
00039 0011          StpBlk
00040 0011 A680             lda   ,X+
00041 0013 8122             cmpa  #'"         quote?.
00042 0015 2706             beq   Skip1
00043 0017 8120             cmpa  #$20        blank?
00044 0019 27F6             beq   StpBlk
00045 001B 301F             leax  -1,X        back up to non-blank
00046 001D          Skip1
00047 001D 9F00             stx   Key
00048 001F 6FA2             clr   ,-Y         mark end of search key
00049 0021          GrepLp
00050 0021 8600             lda   #0          standard in
00051 0023 3044             leax  Line,U
00052 0025 108E0100         ldy   #MAXSIZ
00053 0029 103F08           OS9   I$ReadLn

00054 002C 2511             bcs   Grep1
00055 002E 8D17             bsr   LCompar
00056 0030 25EF             bcs   GrepLp
00057 0032 3044             leax  Line,U
00058 0034 108E0100         ldy   #MAXSIZ
00059 0038 8601             lda   #1          standard out
00060 003A 103F8C           OS9   I$WritLn
00061 003D 24E2             bcc   GrepLp
00062 003F          Grep1
00063 003F C1D3             cmpb  #211        (EOF)
00064 0041 2601             bne   GrepEr
00065 0043 5F               clrb
00066 0044          GrepEr
00067 0044 103F06           OS9   F$Exit
00068
00069 0047          LCompar
00070 0047 9F02             stx   Work
00071 0049 109E00           ldy   Key
00072 004C          LComp2
00073 004C A680             lda   ,X+         from file
00074 004E A1A0             cmpa  ,Y+         from key
00075 0050 27FA             beq   LComp2
00076 0052 6A3F             dec   -1,Y        last char from key
00077 0054 2711             beq   Match       null: A Hit
00078
00079 0056 810D             cmpa  #$0D        (CR) in file?
00080 0058 2700             beq   NoMatch
00081 005A 9E02             ldx   Work
00082 005C 3001             leax  1,X         new starting place
00083 005E 9F02             stx   Work
00084 0060 109E00           ldy   Key
00085 0063 20E7             bra   LComp2      try again
00086 0065          NoMatch
00087 0065 53               comb              set carry
00088 0066 39               rts
00089 0067          Match
00090 0067 5F               clrb              clear carry
00091 0068 39               rts
00092 0069 108C42           emod
00093 006C          srcbsiz equ   *

00000 error(s)
00000 warning(s)
006C 00108 program bytes generated
0284 00516 data bytes allocated
2299 00857 bytes used for symbols
```

MUSTANG-020 Update #3

Several new additions for the MUSTANG-020 68020 Hi-Speed System, from Data-Comp (CPI) have become available in the past month. Deliveries are on time and we are excited about the acceptance of this system. It is performing flawlessly in government, laboratories, schools, industry and of course - serious hackers (me for one). Fact is, we have heard nothing but praise for it. We have three programmers developing C and Sculptor programs on them now, both OS-9 and UniFLEX. Our production cost for software development is going down 2 or 3 fold. As best I can find, the MUSTANG-020 is about $15-20,000.00 less than a comparable system from one of the 'biggies' entry level systems (less RAM, etc.)! Even when the price increases (it will next quarter) it will still be the best bargain around.

Because we have received so... many calls asking about the UniFLEX version, I will try to give you some insight into this super fast operating system.

As many of you might have surmised, from past editorials, I have not always agreed with all of TSC's policies concerning UniFLEX in years past. But having gone into all that before, no need to say more. However, the 68020 version is a "whole new ball-game"! The boys over at TSC have done it right this time. Still not enough basic info (no system managers guide in our latest manual, also no K&R with C, as stated in manual) and no configuration info yet, but I think I heard a little birdie say something awhile back on that subject, so will wait and see. Almost any 6809 UniFLEX, OS-9 or UNIX user will have no trouble, at all. First time users are finding it friendly enough to be 'right in the swing of things' within a short time. We are finding that both the OS-9 and UniFLEX C compilers accept most all UNIX C source without any change. As well as 6809 McCosh C compiler source (FLEX or OS-9). Others adapt with little change. SCULPTOR+ source also compiles from those other systems with little if any change. Not to be left out, BasicO9 and UniFLEX BASIC require little change, if any.

For those with serious questions we are now offering a HANDS ON BEFORE YOU BUY 2 day inhouse trial of both systems. See full page ad elsewhere in this issue.

While on the subject, I have been told that the folks at Microware are doing a major revamp of OS-9 68K, for the 68020 and math co-processor. You might remember that OS-9 68K was done before the 68020 was available. As is, it's an excellent operating system on the 68020. After the overhaul it should be even better. Also I understand that all the HLLe (from Microware) it supports will get the full treatment also. When available it will be ported to the MUSTANG-020.

UniFLEX on the MUSTANG-020 is FAST!! It is heavily optimized for the 68020 and all its extra powerful features, as well as the 68881 math coprocessor. It is simply awesome to see such power from such a small package, as the MUSTANG-020! And to support that power we are introducing several new options, for the OS-9 and UniFLEX versions.

But let me tell you a little about the UniFLEX system first. The support ROM includes a special version of the Motorola 020Bug. This is also included in the OS-9 package, and is a $595.00 value. The system is normally switch configured to 'auto' boot (either floppy or hard disk) on power-on or reset. However, it can be configured to boot from 020Bug, do an auto self-test first and then boot or boot from floppy disk.

Due to the swapping feature (and disk banging of most 6809 systems) this version is nicer to use. Much less, if any disk swapping. Because of the large memory space available (2 megabyte), UniFLEX does memory swapping, on time slices and size modifications. This feature accounts for some of the speed advantage. Also the RAM map is divided into two areas. The system does not use hardware memory management, a software memory management scheme divides the RAM space into two main areas.

The lower 512K bytes of RAM are reserved for user task execution. Each task is therefore limited to a size of 512K bytes. Depending upon the paging "swapping" space left, in RAM, each task can run up to 128 tasks of 512K bytes each (gotta have a lot of RAM OR disk!) simultaneously. The remaining 1.5 megabytes of RAM is depleted by about 200K for system use. Leaving about 1.3 megabytes of RAM for tasks swapping (or paging). If the combined sizes of user tasks exceeds available RAM then disk swapping is done. Therefore, diskettes must still be formatted with a sufficient value of "swap area". The number of tasks is still determined by the amount of swapping space reserved on the hard disk (can't do much on a floppy).

Also available for the UniFLEX version is a new "streaming Tape System". The MUSTANG-020 supports the popular Emulex MT02 streaming tape controller. This controller supports most popular tape streamers. Tape needs must be specified at time of purchase due to special hardware considerations. Call or write for pricing.

Of course the popular 8 port serial RS-232 (db25) expansion is available on the UniFLEX system. All MUSTANG-020 systems can be expanded to 20 serial ports. But for the UniFLEX version there is also a new 9 pin RS-232 interface adapter system. Both boards (25 and 9 pin) provide level shifting between TTL levels and standard RS-232 levels.

THe 9 pin D connectors are so arranged so that "off the shelf" IBM type cables attach directly to standard 25 pin D connectors.

Three of the ports have fixed pinouts (both 9 and 25), arranged so that they can be connected to the standard 0825 protocol. The fourth port (on each four port board) is pin programmable for either DCE or DTE types. The RS-232 inputs and outputs meet RS-232 specs. Each four

port board has a built on de to de level converter. Thus both + and - voltages of the proper values are available.

There will be a continual program of upgrading the MUSTANG-020 in the future. There is a strong possibility that there will be a UNIX System V version before years end. And other important developments underway. So you see, we practice what we have always 'preached'. Quality at a fair price with good support. We require it of ourselves more so than others! And you know what that is!

* * * *

We also have a few 'trade in - 68XXXX' boards and systems left. The list is now worked down, so if you want to get into a 68XXX system at a lesser price, maybe one of these will do it. A small deposit ($50.00) reserves a place on the waiting list. Prices vary from about $6-800 for boards alone to $2,000 for complete hard disk systems, with cabinets and power supplies. Of course, your deposit is refundable at any time before shipping. All trade-ins are tested before we take them in. However, they are sold with no warranty or refund privileges. Also we will, within 6 months of purchasing a 68XXX 'trade-in' from Data-Comp, allow you to apply the full purchase price paid Data-Comp for any 68XXX trade-in in trade-in of that system on a MUSTANG-020, purchased from Data-Comp at prevailing prices, at that time.

DMW

BAD MEMORIES

by Barry Balitski
151 Midglen Place
Calgary, Alberta
Canada T2X 1H6

After the last week I am quite surprised when I look in the mirror and see I still have hair left on my head. My week started normally until I sat down at my computer to write some letters. While editing text I noticed that I appeared to be getting characters on the screen that were not what I thought I had typed. At first I blamed it on my typing, which would make any typing teacher sick with disgust.' It was only upon going through the text and repairing any errors and returning to the same spot and discovering that they had changed again did I really start to realize that I had a serious error of some kind which was changing characters in my text while I worked on editing the file. By carefully examining the errors made such as spaces changed to '(' and 'R's to '2' and consulting the ASCII code chart it became apparent that sometimes bit 3 was being set or cleared whenever it wanted.

O.K...time to pull out the memory tests and see where the problem is. I have several memory tests which were all adapted from the tests supplied with my original SWTPc 6800 system but are now recoded to my FLEX 6809 system. These tests are included at the end of this rambling story. I ran memory tests over the memory one board at a time immediately that night and all memory tests were successful, no erroral About this time my wife became concerned about all the moaning and swearing and came to see if I had been electrocuted as all she could see was my legs sticking out of my SS-50 box. I had a fitfull sleep as memory tests were running through my brain all night. The next day it was back to running tests over each board again with the same results...NO ERRORS !!! I began to think that maybe I had a problem in the memory that FLEX lives in so I wrote a memory test and compiled it in KBASIC to test that memory range...again..NO ERRORS !! My language at this time had my family fearing lightning strikes from above.

In frustration I decided to run the tests over all my available memory in a continuous block. I knew this would be very time consuming..., but over 17 hours I really didn't expect. However, it was worth it as the test uncovered a dual convergence error. The memory test showed that a write to $B9E2 affected memory at location $79E2. This was a real shock to me as these chips aren't even on the same memory board....So the next time you read somewhere to only run tests on one board at a time, BEWARE, the tests may not be telling you the whole truth. It was apparent that this one little chip made of refined sand was responding to a certain combination of inputs and changing bit 3. Once I found the bad chip, replacing it and running tests for another day, confirmed operation was now correct.

The moral of this story is : memory can give any kind of error any time. Run your memory tests often, start them and go to bed. Don't be lulled into a false sense of security because everything seems to be O.K.

I know some documentation says run tests over only one board at a time, but,I suggest you run tests over your entire memory range whenever possible.

Because until this incident I had forgotten how valuable memory tests can be, I enclose three tests I recoded from 6800 to 6809 years ago. Special thanks to SWTPco for providing these with my first computer kit. All of these tests use the I/O routines supplied by the PSYMON monitor which was provided with the now defunct PERCOM SS-50 system. All other monitors will need to change the routine addresses. All of these tests reside in the FLEX utility space to allow you to test the full lower memory range. These tests are also fully position independent and may be put in ROM if desired as all variables are kept on the stack which may be changed from it's present location, now directly at the end of the memory test code.

If anyone has questions or comments they may be addressed to me as indicated at the start of this article.

Editor's Note: These programs are included in 'Reader Service Disk 27'. Please see advertising for these disk elsewhere.

```
*=========================================

*                  ROTABIT

*          ROTATING BIT MEMORY TEST

* THIS DIAGNOSTIC IS A 'WALKING BIT' TYPE. IT
* MOVES  ONE BIT THROUGH EACH BIT POSITION OF
* THE ADDRESS UNDER TEST AND READS BACK  WHAT
* WAS ACTUALLY WRITTEN.
* SUCCESSFUL  PASSES THROUGH THE MEMORY RANGE
* UNDER TEST IS INDICATED BY THE PRINTING  OF
* A  '+' A FAILURE IS INDICATED BY A PRINT OF
* THE ADDRESS, WHAT THE PATTERN  WRITTEN  WAS
* AND  WHAT  WAS  ACTUALLY  CONTAINED  IN THE
* ADDRESS AFTER THE TEST.
* THIS  PROGRAM  WILL  CONTINUE  TESTING  THE
* MEMORY RANGE UNTIL A 'RESET' IS EXECUTED BY
* THE COMPUTER.
* MEANT TO BE USED WITH OTHER DIAGNOSTICS  TO
* FULLY  TEST  MEMORY, AS SOME ERRORS WILL BE
* IGNORED BY THIS PROGRAM.
```

```
*     ROM MONITOR ROUTINES
OUTS    EQU    $FD75      OUTPUT A SPACE
OUT2H   EQU    $FD7D      OUTPUT BYTE (A)
GETADR  EQU    $FD0E      GET HEX ADDRESS (X)
OUTCH   EQU    $FD58      OUTPUT CHAR. (A)
INCHE   EQU    $FD44      INPUT CHARACTER (A)
PDATA   EQU    $FD97      PRINT STRING (X)
PCRLF   EQU    $FDA2      PRINT CR AND LF
EXIT    EQU    $FC32      PSYMON START
DSPDBY  EQU    $FD6A      OUTPUT A&B AS 4 HEX

* LOCAL  STORAGE  EXPRESSED  AS  OFFSETS FROM
* USER STACK ( U-REG). SPACE USED IS RESERVED
* AT THE END OF THIS PROGRAM.

SIZE    EQU    4          BYTES REQUIRED
LOMEM   EQU    0          START ADDRESS
HIMEM   EQU    2          END ADDRESS

        ORG    $C100      POSITION INDEPENDANT

ROBIT   BRA    ROBIT1     BRANCH BY VERSION #
VN      EQU    1          VERSION #

MSG1    FCC    $D,$A,'START ADDRESS ? ',$4
MSG2    FCC    $D,$A,' END  ADDRESS ? ',$4

ROBIT1  LEAU   LOCAL,PCR U-REG LOCAL STORAGE
GETRNG  LEAU   -SIZE,U    SET STACK STORAGE
        LEAX   MSG1,PCR
        JSR    PDATA
        JSR    GETADR     INPUT LOW ADDRESS
        STX    LOMEM,U    SAVE IT
        LEAX   MSG2,PCR

        JSR    PDATA
        JSR    GETADR     INPUT HI ADDRESS
        STX    HIMEM,U
        JSR    PCRLF
START2  LDX    LOMEM,U
LODREG  LDA    #1         STORE 1 (0000 0001)
        STA    0,X        ADDRESS UNDER TEST
        CMPA   0,X        WAS '1' WRITTEN ?
        BNE    ERRPNT     NO,GO REPORT ERROR
LOOP1   ASLA              (0000 0010)
        ASL    0,X        DO TEST ADDRESS
        CMPA   0,X        COMPARE THEM
        BNE    ERRPNT     IF UNEQUAL GO REPORT
        CMPA   #%10000000 FULLY SHIFTED ?
        BNE    LOOP1      NO,CONTINUE ROTATING
        BRA    INCR1      YES,GO TO NEXT
ERRPNT  TFR    X,Y        FAILING ADDRESS
        TFR    A,B        FAILING TEST VALUE
        JSR    PCRLF
        LEAX   0,Y        FAILED ADDR TO X-REG
        JSR    OUT4H      PRINT FAILING ADDR
        JSR    OUTS       AND A SPACE
        TFR    B,A        TEST VALUE TO A-REG
        JSR    OUT2H      PRINT FAILURE ADDR.
        JSR    OUTS       AND A SPACE
        LDA    0,Y        GET DATA TEST ADDR
        JSR    OUT2H      PRINT ACTUAL WRITE
        JSR    PCRLF
        LEAX   0,Y        X TO FAILED ADDR.
INCR1   CMPX   HIMEM,U    END ADDRESS ?
        BEQ    FINISH     BRANCH IF DONE
        LEAX   1,X        NEXT ADDRESS
        BRA    LODREG     AND CONTINUE
FINISH  LDA    #'+
        JSR    OUTCH      SUCCESSFUL PASS
        BRA    START2     CONTINUE TEST
OUT4H   PSHS   D          SAVE A+B FOR LATER
        TFR    X,D        X-REG TO D-REG
        JSR    DSPDBY     OUTPUT A AND B-REG
        PULS   D
        RTS
        RMB    SIZE       USER STACK STORAGE
LOCAL   EQU    *

        END    ROBIT
```

```
*=========================================

*                  SUMSTEST

* THE SUMSTEST MEMORY DIAGNOSTIC WILL PLACE A
* PATTERN IN THE  MEMORY  LOCATION  WHICH  IS
* DEPENDENT  ON  THE  ADDRESS UNDER TEST. THE
* PATTERN WHICH IS WRITTEN IS THE SUM OF  THE
* PASS COUNTER,THE MS BYTE OF THE ADDRESS AND
* THE LS BYTE  OF  THE  ADDRESS.  ERRORS  ARE
* REPORTED  WITH  A PASS NUMBER ,FAILING BITS
* AND THE ADDRESS OF  FAILURE.  THIS  PROGRAM
* RESERVES  SPACE  FOR  STORAGE  ON  THE USER
* STACK AT THE END OF THE PROGRAM IN  'LOCAL'
* STORAGE.  BY RELOCATING THE USER STACK THIS
* PROGRAM MAY BE MADE ROMABLE CODE.

*     EQUATES     PSYMON MONITOR ROM

INCHE   EQU    $FD44      INPUT INTO A-REG
OUTCH   EQU    $FD58      OUTPUT CHAR IN A
PCRLF   EQU    $FDA2      PRINT CR AND LF
PDATA   EQU    $FD97      PRINT STRING (X)
GETADR  EQU    $FD0E      GET 2 HEX IN X-REG
OUT2H   EQU    $FD7D      OUTPUT A AS HEX
OUTS    EQU    $FD75      OUTPUT SPACE
EXIT    EQU    $FC32      PSYMON WARMSTART
DSPDBY  EQU    $FD6A      OUTPUT A&B AS 4 HEX

* LOCAL STORAGE OFFSET FROM USER STACK

SIZE    EQU    8          BYTES REQ'D
CTR     EQU    0
STORE   EQU    1
TEMPX   EQU    2
LOMEM   EQU    4
HIMEM   EQU    6

        ORG    $C100      P.I.C. & ROMABLE

SUMTST  BRA    SUM1       BRANCH VERSION #
VN      EQU    1          VERSION #

MSG1    FCC    $D,$A,'STARTING ADDRESS ? ',$4
MSG2    FCC    $D,$A,' ENDING  ADDRESS ? ',$4

SUM1    LEAU   LOCAL,PCR LOCAL STORAGE
        LEAU   -SIZE,U    SET ASIDE AREA
        CLR    STORE,U
        CLR    CTR,U
        LEAX   MSG1,PCR
        JSR    PDATA
        JSR    GETADR     GET START ADDRESS
        TFR    X,Y        Y=START ADDRESS
        LEAX   MSG2,PCR
        JSR    PDATA
        JSR    GETADR     GET END ADDRESS
        JSR    PCRLF
        LEAX   1,X
        STX    HIMEM,U
        STY    LOMEM,U
START   TFR    Y,X        START ADDRESS INTO X
LOOP1   BSR    INCRX
        STA    0,X        X= TEST ADDRESS
        LEAX   1,X        BUMP TO NEXT ADDRESS
        CMPX   HIMEM,U    END OF RANGE YET ?
        BNE    LOOP1      NO CONTINUE
        LDX    LOMEM,U
LOOP2   BSR    INCRX      FORM DATA AGAIN

        EORA   0,X        DIFFERENT BITS SET
        BNE    ERROR      NO,REPORT ERROR

RETURN  LEAX   1,X        YES,BUMP UP
        CMPX   HIMEM,U
        BNE    LOOP2
        LDA    #'+
        JSR    OUTCH
        INC    CTR,U      SUCCESSFUL PASS
        JSR    EXITST     TEST FOR END
        BRA    START      GO DO IT AGAIN
```

```
INCRX   STX     TEMPX,U     SAVE TEST ADDR
        LDA     TEMPX,U     MSB TO A-REG
        ADDA    TEMPX+1,U   ADD LS BYTE
        ADDA    CTR,U       ADD PASS #
        RTS
ERROR   STA     STORE,U     SAVE ERRANT BIT
        JSR     PCRLF
        LDA     CTR,U
        JSR     OUT2H       PRINT PASS #
        JSR     OUTS        SPACE
        LDA     STORE,U
        JSR     OUT2H       PRINT FAILING BIT
        JSR     OUTS        SPACE
        LDX     TEMPX,U
        JSR     OUT4H       PRINT FAILING ADDR
        JSR     OUTS
        BRA     RETURN
EXTST   RTS                 STUB FOR EXIT TEST
OUT4H   PSHS    D           SAVE A+B FOR LATER
        TFR     X,D         X-REG TO D-REG
        JSR     DSPDBY      OUTPUT A&B REG.S
        PULS    D
        RTS

        RMB     SIZE        USER STACK STORAGE
LOCAL   EQU     *

        END     SUMSTEST

*-----------------------------------------------

*                   CONDATA

*         CONVERGING DATA MEMORY TEST
* THIS DIAGNOSTIC TESTS FOR 'DUAL ADDRESS
* ERRORS'. IT TESTS EACH LOCATION OF THE
* RANGE UNDER TEST BY FIRST WRITING ALL 0's
* TO EACH LOCATION , THEN WRITING ALL ONES TO
* THE FIRST LOCATION AND CHECKING ALL OTHER
* LOCATIONS FOR ANY SET BITS IN THE TEST
* RANGE. IF NO BITS ARE SET THEN THE 'ONES'
* ARE REPLACED WITH ZEROES THEN THE NEXT
* LOCATION IS FILLED WITH THE 'ONES' OR FF
* PATTERN AND ALL OTHER LOCATIONS ARE AGAIN
* CHECKED UP TO THE END OF TEST RANGE.

* THERE ARE FOUR TYPES OF ERRORS THAT ARE
* REPORTED VIA THE SWI (SOFTWARE INTERRUPT)
* INSTRUCTION , THE ADDRESS OF FAILURE AND
* TYPE ARE REPORTED IN THE CPU REGISTERS ON A
* REGISTER DUMP. A PASSING CONDITION IS ALSO
* REPORTED.

*   ON AN ERROR THE REGISTERS WILL BE DUMPED
*   X-REG CONTAINS THE FAILED ADDRESS
*   PC-REG CONTAINS ADDRESS OF ERROR TYPE
*         (SEE LISTING)
*   IF DUAL ADDRESS ERROR
*   X-REG CONTAINS THE FAILED ADDRESS
*   Y-REG CONTAINS AFFECTED ADDRESS

*   APPROX TIME AT 1 MHZ.
*   1K                24 SEC.
*   2K                 2 MIN.
*   4K               7.5 MIN
*   8K                30 MIN.
*  16K                 2  HR.
*  48K                17 HRS.

* System equates     PSYMON rom.
PDATA   EQU     $FD97       PRINT STRING (X)
GETADR  EQU     $FD0E       GET ADDRESS IN X
PCRLF   EQU     $FDA2       PRINT CR & LF
EXIT    EQU     $FC32       PSYMON WARMSTART

* Local storage expressed as
* offsets from (U) Stack
SIZE    EQU     4
LOMEM   EQU     0
HIMEM   EQU     2

        ORG     $C100       POSITION INDEPENDENT

TEST    BRA     TEST1       BRANCH VERSION #
VN      FCB     11          VERSION #

MSG1    FCC     $D,$A,'Start Address ?',4
MSG2    FCC     $D,$A,'End Address ?',4
MSG3    FCC     $D,$A,'in progress...',4
MSG4    FCC     $D,$A,'Memory test OK ',$7,4

**   USER STACK ALTER DEPENDING ON SYSTEM
TEST1   LEAU    LOCAL,PCR USER STACK
GETRNG  LEAU    -SIZE,U     SET UP STORAGE
        LEAX    MSG1,PCR
        JSR     PDATA
        JSR     GETADR      GET START ADDRESS
        STX     LOMEM,U     SAVE IT FOR LATER
        LEAX    MSG2,PCR
        JSR     PDATA
        JSR     GETADR      GET END ADDRESS
        STX     HIMEM,U     SAVE IT
        LEAX    MSG3,PCR
        JSR     PDATA       PRINT "IN PROGRESS .
START   LDX     LOMEM,U
        CLRA                PUT 00 INTO ACC. A
LOOP1   STA     0,X         A INTO TEST ADDRESS
        CMPA    0,X         WAS 00 WRITTEN?
        BNE     ERPNT1      NO GO REPORT ERROR 1
        CMPX    HIMEM,U     END OF RANGE YET ?
        BEQ     LOAPAT      YES NEXT PATTERN
        LEAX    1,X
        BRA     LOOP1       NO CONTINUE FILL
LOAPAT  LDX     LOMEM,U
        LDB     #$FF        SECOND PATTERN  (FF)
LOOP4   STB     0,X
        CMPB    0,X         WRITTEN CORRECTLY ?
        BNE     ERPNT2      NO REPORT ERROR 2
CHKLOW  TFR     X,Y         SAVE TEST ADDRESS
LOOP2   CMPX    LOMEM,U     FIRST ADDRESS ?
        BEQ     CHCKHI
        LEAX    -1,X
        CMPA    0,X
        BNE     ERPNT3
        BRA     LOOP2
CHCKHI  TFR     Y,X
        CMPX    HIMEM,U
        BEQ     FINISH
LOOP3   LEAX    1,X
        CMPA    0,X
        BNE     ERPNT4
        CMPX    HIMEM,U
        BNE     LOOP3
RESTRE  TFR     Y,X
        STA     0,X
        LEAX    1,X
        BRA     LOOP4

*     ERROR BLOCK
*   ALL SECTIONS RESTORE USER STACK

ERPNT1  LEAU    SIZE,U
        SWI                 ERROR INITIAL '00'
ERPNT2  LEAU    SIZE,U
        SWI                 ERROR PATTERN 'FF'
ERPNT3  LEAU    SIZE,U
        SWI                 DUAL ADDRESS   LOW
ERPNT4  LEAU    SIZE,U
        SWI                 DUAL ADDRESS   HIGH

FINISH  LEAX    MSG4,PCR    SUCCESSFUL MESSAGE
        JSR     PDATA
        LEAU    SIZE,U      RESTORE STACK
        JMP     EXIT        TO MONITOR ROM
        RMB     4           STORE LO & HI MEM
LOCAL   FCB     0           STORAGE USER STACK
        END     TEST

*-----------------------------------------------
```

ACCESSING
The FLEX DIRECTORY
From BASIC

ACCESSING THE FLEX DIRECTORY
FROM BASIC
R. H. Hoglund, Seattle WA

I have often wanted to have directory information available from FLEX to process with an XBASIC program. The string handling features of BASIC can be extremely useful in creating means to delete or perform other functions at the OS level on a selective basis.

The following program shows the technique, and also is useful itself. I had a need to selectively delete data files that multiply like rabbits on a data disk that captures stock market statistics from the Compuserve data base. Although this can be done in assembler, the attached listing in BASIC shows how easily it can be done in higher level language.

The SDEL program has a good feature of the ERASE command in CPM and later OS's. It allows a limited wild card specification of the file names to be displayed, and deleted or kept at the operators discretion. The leading character(s) of the filename will initiate a search for all names starting with that string. In addition, an asterisk can be used for either the filename or extension to select all possible names or extensions. To illustrate, if the following were typed in in answer to the prompts, then the result would be just like the PDEL utility:

```
EXTENSION OR STAR FOR ALL? *
LEAD CHAR OF FILENAME OR STAR? *
```

As another example, if "P" were given in answer to the request for the lead character, then all files beginning with P would be displayed.

The trick to getting XBASIC to do this, is to transfer the disk directory information to a form (file) readable by BASIC. The directory format selected was that of the FILES command in the utilities package, where only the filenames are displayed on the screen. This is easily turned into a file with the "O" command in Flex. When the program is run, the scratch file should not already exist. This results in an error #4, which is trapped by the ON ERROR condition in line 8. Line 38 determines that this is a "no such file" error, and branches to line 49.

At line 51, the EXEC command causes the output of the FILES command to be written to a new file, DIRSCR.SCR. This works fine, but when two utilities are called on the same line (O and FILES in this case), the vectors in Flex are reset leaving FILES to return to the O command (calling) utility, and the EXEC sequence returns to Flex instead of to BASIC. The vectors can be redirected to BASIC by the unusual step of chaining the program to itself in line 53.

The second time the program runs (as a result of the chain) the input file now exists, and the main body of the program executes. The loop at line 9 gets rid of the header generated by FILES. Each line in the directory information is scanned for successive period characters between a filename and its extension by line 18. The 3 characters following the period are assumed to be the extension, and the filename built up in variable FILE_NAME$ in this way.

The filenames of interest are displayed one at a time, and the operator asked if they should be deleted. The program loops until end-of-file is detected.

Line 19 is of interest. It detects a condition where a line contains no more periods, and then calls for another line of data from the file. Line 32 seems to have the same function, but does not work because of trailing spaces on each line from the file. It was left in as a safety measure.

The final step is on end-of-file, and after the procedure is complete. The flow will reach lines 37 to 53. Notice that the scratch file is deleted in line 44. This is necessary for two reasons. First, the file must be absent the next time the program is run to trigger the EXEC command. Second, if any file is deleted during the use of the program, any attempt to rerun the program will make use of the old scratch file, and when the program can't find the file that was previously deleted, an error will result, with unpredictable (but not catastrophic) results due to the error trapping used.

The BASIC program is written for the TSC XPC pre-compiler to make it easier to follow the program steps. By substituting two character variable names, and using line numbers instead of statement names, it can also be compiled in XBASIC with the COMPILE command. It should be stored in compiled form to speed up the operation of the CHAIN statement.

```
2   @TTL SDEL.TXT/.BAC
3   @REMOVE SELECTED FILES FROM DIRECTORY
4   @THIS VERSION REMOVES FILES USING LIMITED WILD CARD
5   @DISK 143 OR 1013
6   *
7               OPEN OLD "I.DIRSCR.SCR" AS 2
8               ON ERROR GOTO ERRECOV
9                    FOR I% = 1 TO 3  ; REM SKIP HEADER INFO
10                   INPUT LINE #2,LLINE$
11                   NEXT I%
12              INPUT"EXTENSION OR STAR FOR ALL",EXTEN$
13              INPUT"LEAD CHAR OF FILENAME OR STAR",LEAD=CHAR$
14              SRCH=LEN%=LEN(LEAD=CHAR$)
15  MAIN=LOOP   LPOINTER% = 1
16              INPUT LINE #2,LLINE$
17              LINE=LEN% = LEN(LLINE$)
18  INNER=LOOP  PERIOD=LCN% = INSTR(LPOINTER%,LLINE$,"-")
19              IF PERIOD=LCN% = 0 THEN MAIN=LOOP ; REM DETECT END OF LINE
20              FILE=NAME$ = ""  ; REM SET TO NULL
21                   FOR I% = LPOINTER% TO PERIOD=LCN% + 3
22                   INDIV=CHAR$ = MID$(LLINE$,I%,1)
23                   IF INDIV=CHAR$ <> " " THEN {
                      FILE=NAME$ = FILE=NAME$ + INDIV=CHAR$
24                   NEXT I%
25              IF EXTEN$ = "O" THEN  {
                 IF LEFT$(FILE=NAME$,SRCH=LEN%) <> LEAD=CHAR$ THEN SKIP=DEL
26              IF LEAD=CHAR$ = "O" THEN I
```

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

K-BASIC now makes the multitude of TSC HBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), including the Assembler

!!! Special !!! ~~$199.00~~ $ 99.49

SAVE $100.00

SCULPTOR

Features

THE SCULPTOR SYSTEM
Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER
Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE
Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY
Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and -- without any alterations to the programs -- run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

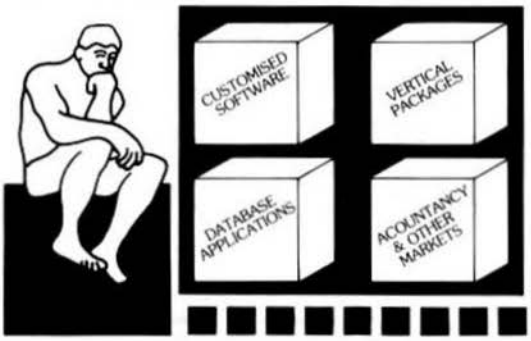
SPEED AND EFFICIENCY
Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED
By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe -- Sculptor is already at work in over 20 countries.

THE PACKAGE
With every development system you receive:
- ☐ A manual that makes sense
- ☐ A periodic newsletter
- ☐ Screen form language
- ☐ Report generator
- ☐ Menu system
- ☐ Query facility
- ☐ Set of utility programs
- ☐ Sample programs

For resale products, the run-time system is available at a nominal cost.

FIND OUT MORE



DATA DICTIONARY
Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:
- ☐ alphanumeric
- ☐ integer
- ☐ floating point
- ☐ money
- ☐ date

DATA FILE STRUCTURE
- ☐ Packed, fixed-length records
- ☐ Money stored in lower currency unit
- ☐ Dates stored as integer day numbers

INDEXING TECHNIQUE
Sculptor maintains a B tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION
Input data may be validated at three levels:
- ☐ automatic by field type
- ☐ validation list in data dictionary
- ☐ programmer coded logic

ARITHMETIC OPERATORS

-	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
-	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
and	Logical and
or	Logical or
cf	Concerns
bw	Begins with

SPECIAL FEATURES
- ☐ Full date arithmetic
- ☐ Echo expression for commands
- ☐ Terminal and printer independence
- ☐ Parameter passing to sub-programs
- ☐ User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS
- ☐ Define record layout
- ☐ Create new indexed file
- ☐ Generate standard screen-form program
- ☐ Generate standard report program
- ☐ Compile screen-form program
- ☐ Compile report program
- ☐ Screen-form program interpreter
- ☐ Report program interpreter
- ☐ Menu interpreter
- ☐ Query facility
- ☐ Reformat file
- ☐ Check file integrity
- ☐ Rebuild index
- ☐ Alter language and date format
- ☐ Setup terminal characteristics
- ☐ Setup printer characteristics

SCREEN-FORM LANGUAGE
- ☐ Programmer defined options and logic
- ☐ Multiple files open in one program
- ☐ Default or programmer processing of exception conditions
- ☐ Powerful verbs for input, display and file access
- ☐ Simultaneous display of multiple records
- ☐ Facility to call sub-programs and operating system commands
- ☐ Conditional statements
- ☐ Subroutines
- ☐ Independent of terminal type

Sculptor for 68020 OS-9 and UniFLEX $995

OS-9 / UniFLEX --
IBM PC Zenix -- $995.00/$175.00
MS DOS Network -- * **

68000 UniFLEX --
Altos Zenix -- $1595.00/$265.00
UNIX -- * **

MS DOS --
-- $595.00/$115.00
PC DOS -- * **

* Full Developement Package ** Run Time Package Only

Full OEM and Dealer Discounts Available!

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the TSC **Macro Assembler**. F, CCF - $99.95

Macro Assembler for TSC -- The FLEX **STANDARD** Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from **TSC**. -- Use with many of the C and Pascal Compilers. F,CCF $150.00

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - $98.00

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- (REAL ASSEMBLERS, NOT MACRO SETS) Specify for 180x, 6502, 6801, 6804, 6805, 6809, Z8, Z80, 8048, 8051, 8085, 68000 modular, free-standing cross-assemblers in C, with load/unload utilities and macros.
FLEX, CCF, OS-9, UniFLEX:
Each $50.00 or Any 3 for $100.00 or ALL $200.00
8 bit (not 68000) sources for additional:
Each $50.00 or Any 3 for $100.00 or ALL $300.00

XASM Cross Assemblers for FLEX from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC **Macro Assembler** Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - $150.00

CRASM8 from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- $399.00

CRASM 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 $249.00

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCD, Obj. Only $50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide. F, CCF - $198.00

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - $195.00

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - $295.00

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, CCF, OS-9 (Level II ONLY), U - $575.00

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - $99.95 F 8" $99.95

PASCAL Compiler from **CompuSoft** (now **Certified Software**) -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Assmb. and Linking Loader.
F and CCF - $425.00 One Year Maint. - $100.00

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - $199.00

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- $99.95

FORTH from **Staaras Electronics** -- A CoCo FORTH Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, **Trace**, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer ONLY - $98.95

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

BasicO9 XRef from Southeast Media -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCF -- **XREF Utility** 5" - $40.00, 8" - $50.00

DU8 from Southeast Media -- A UniFLEX "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled basic** Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 VDisk from Southeast Media -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in **BASICO9** (with Source), includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an **OS-9** disk to **FLEX Format** so it can be used normally by FLEX; and **FLEX**, a BASICO9 Program that does the actual **read** or **write** function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** O-F $39.95

COPYMULT from Southeast Media -- Copy **LARGE** Disks to several **smaller** disks. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal NOT required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - $50.00 F 8" - $65.00

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test** Disk for errors; **Compare** two **Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with **EXTRAS** over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare**, **Merge**, or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare **two unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC **Cross-Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC**, **XBASIC**, and **COMPUTIST** BASIC Programs.
ALL Utilities **include Source** (either BASIC or A.L. Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for **UniFLEX** -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $90.00

XDATA from Southeast Media -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 Chess Program from Southeast Media -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

Telex 5106006630

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

BUSINESS

WORD PROCESSING

SCREDITOR III from **Windrush Micro Systems** -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from **Great Plains Computer Co.** -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES —> CCF and CCU - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from **Great Plains Computer Co.** -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES —> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from **Great Plains Computer Co.** -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES —> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from **Southeast Media** -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the **FPRINT.CMD** supplied for producing multiple copies of the "Formatted" Text on the Printer **INCLUDING IMBEDDED PRINTER COMMANDS** (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: **FLEX** - CC
Disk #2: JUSTSC object and source in C: **FLEX** - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax **JUST.CMD** object file. Using JUST syntax (.p ,u ,y etc.) With all JUST functions plus several additional printer formatting functions. Reference the **JUSTSC** C source. For those wanting an excellent **BUDGET PRICED** word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - **$49.95**
Disk Set (2) - F & CCF & OS9 (C version) - **$69.95**

SPELL8 "Computer Dictionary" from **Southeast Media** -- **OVER 150,000 words!** Look up a word from within your Editor or Word Processor (with the **SP8.CMD** Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of Small Disk Storage systems.

!! SPECIAL LIMITED TIME OFFER !! F and CCF - **$99.95**

DATA BASE - ACCOUNTING

XDMS from **Westchester Applied Business Systems** -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc. XDMS Level IV was Westchester Applied Business Systems ad this issue.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95
Upgrades to Lvl IV - $250.00
XDMS Lvl IV - F & CCF - $350.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from **Computer Systems Consultants** -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's **Extended BASIC.**

F and CCF, U - **$50.00**, w/ Source - **$100.00**

DYNACALC from **Computer Systems Center** -- Electronic Spread Sheet for the 6809.

F, **SPECIAL** CCF and OS9 - **$200.00.** U - **$395.00**

FULL SCREEN INVENTORY/MRP from **Computer Systems Consultants** -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's **Extended BASIC.**

F and CCF, U - **$50.00**, w/ Source - **$100.00**

FULL SCREEN MAILING LIST from **Computer Systems Consultants** -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's **Extended BASIC.**

F and CCF, U - **$50.00**, w/ Source - **$100.00**

DIET-TRAC Forecaster from **Southeast Media** -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - **$59.95**, U - **$89.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $1.50)
Add 5% Surface Foreign
10% Air Foreign

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

SOUTH EAST MEDIA
5900 Cassandra Smith Rd. CoCo OS-9™ FLEX™
Hixson, TN 37343
mlo (615) 842-4601
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

```
            IF RIGHT$(FILE$+NAME$,3) <> EXTEN$ THEN SKIP-DEL
            IF EXTEN$ <> '.' AND LEAD-CHAR$ <> '.' THEN &
             IF LEFT$(FILE$+NAME$,SRCH-LEN%) <> LEAD-CHAR$ &
             OR RIGHT$(FILE$+NAME$,3) <> EXTEN$ THEN SKIP-DEL
                PRINT'DELETE FILE 'FILE$+NAME$'
                INPUT LINE YESNO$
                IF YESNO$ = 'Y' THEN  KILL FILE$+NAME$
SKIP-DEL    LPOINTER% = PERIOD-LEN% + 4
            IF LPOINTER% >= LINE-LEN% THEN MAIN-LOOP
            GOTO INNER-LOOP
*
*ERROR RECOVERY SECTION
*
ERRECOV     IF ERR <> 0 AND ERR <> 4  &
             THEN ON ERROR GOTO
            IF ERR = 4 THEN FILE-SETUP
*
*EXIT SEQUENCE
```

```
*
            RESUME CONT1
CONT1       CLOSE 2
            KILL '1.DIRSCR.SCR'
            END
*
*CREATE SCRATCH FILE
*
FILE-SETUP RESUME CONT2
CONT2           CLOSE 2
                EXEC.'D.1.DIRSCR.SCR,FILE1'
                PRINT 'SCRATCH FILE BUILT'
                CHAIN '0.SDEL.BAC'
            END


TOTAL ERROR% = 0
```

Omegasoft Pascal OS-9 / 68020

I have had a MUSTANG-020 computer for just about a month as I write this. My first priority was to get familiar with OS-9 and to do some things with Microware "C". The project of translating PAT from PL/9 to C and getting it running has accomplished both goals nicely for me, so the next order of business was to check out the OmegaSoft Pascal compiler that CERTIFIED SOFTWARE sent to 68 Micro Journal for review. The compiler comes on SEVEN 5 1/4" disks, and I was not looking forward to the installation process. However, the instructions were clear and all was accomplished very quickly. I then compiled the test program that is supplied with the system (a Prime Sieve program, of course). I was favorably impressed with the quickness of the compile and even more favorably impressed with the speed of assembly and linking.

Let me back up a moment here. About 5 years ago, I used OmegaSoft Pascal FLEX 6809 version for a couple of years to develop some software for a multi-axis servo control system. My chunk of the program was about 32K of object code. At that time I became quite familiar with the product and the procedures for compiling programs. However, I hadn't used OMEGASOFT Pascal for three years, and I had to brush up on it a little. After getting the test program going, I began to search my 6809 system files for appropriate programs to compile and run as tests. A few years ago I did a benchmark program for compiler evaluation, and I had the Pascal version plus the test results for the 6809 system at 1 MHz, which is the system that I had at the time. I typed in the benchmark program and found all my typos by compiling the program a number of times until I had no errors. I ran it and I will compare the results for the 6809 system with those for the MUSTANG-020. Be aware that the 6809 results are for a 1 MHz system, and that all the times can be divided by 2 for a comparison with the 2 MHz systems that most of us run now

	6809	68020
Compile Time	142 sec.	16 sec *
Object code bytes	4300	5840
Sieve Program	28 sec	1.06
Array Assignment	9.0	0.55
Square Root	9.2	0.43
Integer Mult Div	7.0	0.13
Real Mult Div	8.0	0.47
Parameter passing	4.5	0.22.

The 16 second compile time for the 68020 was obtained using Ramdisk for the intermediate files, but with the compiler and the program files on the hard disk. I expect from other tests that the compile time using Ramdisk only, would be around 8 seconds. I was pleasantly surprised that the object code was only about 25% larger than the 6809 version. Obviously the '020 has instructions that are on the average quite a bit longer than the '09 instructions. Apparently, those instructions are also more capable, so that fewer of them are required to do the same job.

The largest increase in performance is obviously in the Integer arithmetic area where the 68020 version is about 50 times faster, but the gain of 16 times for Real arithmetic is also impressive.

So much for performance. How about all the other things that make or break a language / compiler's usefulness? Pascal, the language, is THE standard for source code readability in my opinion (and that of many other people). You have to be very explicit about what you want Pascal to do, therefore someone else who reads the code will find it all spelled out nicely. Certainly Pascal is a little more verbose than "C" at the source level, but that goes along with the improved readability.

The documentation that comes with this package is not skimpy. It is supplied with a plastic binder but better fits a three ring looseleaf notebook. The pages are printed on both sides and the stack is nearly an inch thick. The various sections of the manual are nicely indexed with tabbed dividers. A working knowledge of how relocatable assemblers and linking loaders work would be helpful to understanding all the steps in getting from the source code to an object program, but the process is pretty well automated by the "lc" utility. "lc" is a "Linkage Calculator" that makes it simple to use the compiler. First you use the "pc" (Pascal Compiler) to read your source code and generate an assembler source file. Then you type "lc" and answer the questions that are asked of you. Generally the answer is Y or N or perhaps the desired stack size. Several of the questions may simply be answered with a CR if no information is required. When you exit the lc utility you will find several new files on your disk. One is a .ps file which is linked to your program. It defines the connection to the operating system, i.e. execution of the program and return to the OS at the end. A second is the .ps file which sets up the stack allocation according to the information you entered into the Linkage Calculator. Another is a .cf file (Chain File) which is the command file to do all the remaining steps in compiling the program.

The Chain File need be made only once. That is you only need to use lc once for a program. Having the .cf file, you can recompile the program over and over again with two commands: pc -o file . will compile the file "file" to an assembler source file with the name "file.co".

It returns to the OS when done. Now, having used lc to get the chain file simply type "file.cf" and watch everything happen automatically and very quickly. When it is done, you will have a "file.lo" in the current directory. This is the executable object file. You can move it to the commands file or the current execution directory, or run it from its present location.

The compiler has options for including debug information in the compiled output, for including the source lines as comments in the compiler output, etc. The debugger allows you to step through a program, print values of variables at various points, set breakpoints etc.

One of the best things about this package is that virtually all of the runtime routines are supplied in assembler source form. If you are just compiling and running Pascal programs on the system, you can ignore the source code, but if you are developing software for a stand alone system, you can write your own I/O module, supply assembler procedures, etc.

OmegaSoft Pascal complies with the Pascal standard very well, but it also includes a number of extensions that make it vastly more useful in development of stand alone software. The most useful of these is the ability to declare a variable "at" an absolute address. You can access an I/O port quite easily using this feature.

I should also mention that it has multiple sizes of each of the variable types. INTEGER is a 16 bit integer. LONGINTEGERs are 32 bits. HEX is 16 bits, LONGHEX is 32. REALs are 32 bits and LONGREAL 64 bits. A variable type STRING is also supported, that is more useful than the normal Pascal array of char.

Let me stress that the extensions to Pascal need not be used, nor do they displace any standard Pascal syntax or operation.

Sorting

A few years ago I wrote a book on Pascal for TAB. The title is "From BASIC to PASCAL". In it I have a chapter on sorting, that includes a number of programs that I had set up to sort 1000 integers for comparison of the relative merits of various sorting algorithms. I had run these in Lucidata Pascal, and tested them in OmegaSoft Pascal in the process of writing the book. I copied some of these to the MUSTANG-020 system and found that I had to make one change to make them compatible with this Pascal package. I had declared my data file of random numbers as DATA : FILE OF CHAR; OMEGASOFT Pascal wouldn't buy that. I immediately assumed that it had a predeclared type TEXT : declared as FILE OF CHAR, so I changed the declaration to DATA : TEXT; Pascal bought that, and the programs compiled. Of all the sorts I tested, the old simple Bubble sort was the slowest. On the old 1 MHz 6809 system it ran some 600 seconds to sort 1000 integers (in 6809 OMEGASOFT Pascal version 1.0). I have not tried the test in the newer versions of 6809 OMEGASOFT Pascal. Anyway, the 68020 version ran just about precisely 40 seconds or 15 times faster.

On the more efficient end of sorting algorithms there are the Shell Metzner sort and the Quicksort algorithm. The 6809 OMEGASOFT Pascal versions sorted the 1000 items in 16 seconds and 8 seconds respectively. The 68020 versions maintained the same 15 times speed up, running the two in 1 second and 1/2 second respectively as nearly as I could determine. It is difficult to increase the number of items in the sort and extrapolate the time, though I could have done the sort ten times (which would necessitate reading the unsorted data into the array ten times also), and divided the total by ten to get a better feel for the time improvement. I suppose the actual and accurate time is largely irrelevant, the main point being the 15 or 16 times speed factor gain.

A Parallel

I see a parallel here. The 68xxx line has been around for some time. The initial reports of execution times using the 68008 were disappointing. The times were not much better (if at all) than the 6809s which by that time were running at 2 MHz.

I remember, when the 6809 first appeared, everyone simply re-assembled all their old 6800 software products with the '09 assembler, and not too surprisingly, the performance improvement was not very exciting. Then over a period of two or three years, the suppliers started writing true 6809 code that took advantage of the improvements (the MUL instruction, for example) and we began to see a decade or so improvement in execution time. Now that the 68xxx is finally getting off the ground (I see lots of software available or being worked on), we are seeing the performance improvement of which the chip is capable. Obviously the 68020 is more capable than the 68000 and that chip is more capable than the 68008, but I would hazard a guess that at least part of the improvement is due to the generation of some code that takes full advantage of the capabilities of the 68xxx instruction set.

One Objection

Those of you who have been reading my column for a long time know me better than to think I could review something without at least one negative comment or suggestion. The Compiler produces good error messages but they are far too cluttered. I'll attempt to reproduce a compiler output that resulted from the omission of a close subscript (]) in an array declaration in a short program.

```
** 28 ] expected
                                  *V*
Compiler action : Compilation resumed at ;
  19:3   iaray : array [1..1000 of integer;
** 72 Statement expected
                                  *V*
Compiler action : Compilation resumed at ;
  26^4   for account := 1 to 1000 do iarray[account] :=
account;
```

The " ** 28 etc. " is the error message. The "*V*" points at the place where the error occurs. My problem with the report is that there need to be blank lines separating the error messages. I could well do without the Compiler action line, which would put the pointer directly above the error spot in the program line. The message conveys no useful information to me to help me understand the error, and it gets in the way of the pointer. Actually most compilers print the line with the error first and then print an up arrow on the next line below to point at the error position. This compiler has done it the Motorola way. Motorola's early CORES assembler is about the only one I know of that always put the error message BEFORE the offending line. The downward pointing V is not the problem in this case, however. It is the line between the pointer and the program line that clutters things up. To my eye, it is very hard to separate the error reports visually also.

Conclusion

At its current price of $900, I am sure that not too many of you are going to rush out and buy OmegaSoft Pascal. On the other hand, if you are developing software for any sort of stand alone system (or maybe even application software to run on the 020 under OS-9) you need this package. I didn't mention this above, but the benchmark program generated 5480 bytes in Pascal, and 16170 bytes in Microware C. Efficiency for small programs should be quite good. I must also mention here that I was one of the very early customers of OmegaSoft with the 6809 Pascal, and that the customer support was excellent right from the start. We did find a few bugs, but they were fixed very quickly. The fact that OmegaSoft has been around now for several years should provide some confidence that they will be around for some time to maintain and improve their software. I can do no less than heartily recommend this compiler for serious software developers. Bob Reimiller, you are to be commended on this package. It is excellent in performance. You must have taken advantage of the performance of the 68020 in every possible place, to have produced such tight code and high speed execution!

Editor's Note: This compiler, as is most all 680XX software, available from S. E. MEDIA, and for Mustang-020 owners and 68 Micro Journal subscribers a discount is available.
This is one of the better software development packages for the 680XX. We hear nothing but good words for Certified Software and their OMEGASOFT PASCAL compiler. It is real "Industrial Strength"!

DMW

- - -

Low Cost Program Kits

The following is a set of programs to be sold as S.E. MEDIA Special Value Kits. It is the desire to keep prices low and furnish source.

These programs will be available on 5 or 8 inch disk, please specify size disk (5 or 8) when ordering.

A listing of parts of the documentation is published here in order that you might have some idea of the disks or programs available under this special program.

Disk numbers 2A and 2B are sold ONLY as a set of two (2) disks when requested on 5".

It is hoped that others will make available their software for inclusion in the S.E. MEDIA "Baargain Hunters Program Corner", or the Special Value Kits offerings.

Many of these programs would sell for much more, but S.E. MEDIA hopes to continue to make affordable software available to our readers through these programs.

If you belive that you have something to offer for this type market, and want to pick up a little change also - please contact S.E. MEDIA and let them know.

PRICES ARE:

Disk #1	Basic Tool-Chest	$29.95
Disk #2	Flex Utilities Kit	$39.95
Disk #3	Assemblers & Disassemblers Utilities	$39.95
Disk #4	Word-Processor Support Utilities	$49.95
Disk #5	Utilities for Indexing	$49.95

NOTE: Every program has the source included, and the documentstion is in text files on each disk.

If you think that S.E. MEDIA should continue this sort of procurement program, please let them know.

DMW

DISK # 1
BASIC TOOL-CHEST
$ 29.95

BLISTER

BLISTER is a flex utility used to produce a formatted listing of a T.S.C. BASIC program from disk to either the terminal, a printer, or a disk file (using the Flex O.CMD).

All multiple statements on a line can be optionally outputted on separate lines, giving a clearer listing. Furthermore all FOR/NEXT loops can be optionally incrementally indented giving a clear indication of nesting.

REMPAC, SPCPAC, COMPAC

The three programs, written in TSC Extended Basic, perform the following functions:

REMPAC: removes all REMARK statements in a Basic file. (See note under COMPAC).
SPCPAC: removes all unnecessary spaces in a Basic file, except those within quotes.

COMPAC: removes all REMARK statements AND unnecessary spaces. Thus it performs the functions of both REMPAC and SPCPAC.

*** Note ***
If a REM statement is referenced from another line, the resulting program will give a runtime error #60, (line not found).

Output may be directed to the terminal, printer or disk.

STRIP.BAS

This program, also for TSC XBasic, removes all UNREFERENCED line numbers from a Basic file. At the same time all referenced line numbers may be OPTIONALLY prefixed with a string of 1 or more characters.

A further option is to generate a disk file of the referenced line numbers in a form suitable for the CHGNAM.CMD so that the latter may be used to convert the numbers into meaningful labels. The output format is: delimiter,line number,delimiter. For example: /2400/. Using CHGNAM this could be set to /2400/SORTARRAY/ so that every occurrence of the line number 2400 would be converted to SORTARRAY.

The program is useful to convert ordinary Basic files to files suitable for the TSC Precompiler, which does not need line numbers on every line and referenced lines can be labelled with a string.

In conjunction with the Flex-to-OS9 file transfer utility OF, it can convert a TSC XBasic file to one compatible with Microware's BASIC09 (after some editing), which also does not require line numbers on every line.

Output may be directed to the terminal, printer or disk.

(* Program available from Southeast Media.)

LINEXREF.BAS

Produces a cross-referenced listing of either all the lines in an XBASIC program or of just those lines that are destinations.
Output may be directed to the terminal, printer or disk.

$ 39.95

CATS

CATS command is used to display a SORTED list of the FLEX disk filenames in the directory of the disk by filename or extension.

CATD

CATD command is used to display a DATE-SORTED list of the FLEX disk filenames in the directory of the disk.

COPYSORT & COPYDATE

COPYSORT allows a disk to be copied to another disk with the files copied over in alphabetical order.

COPYDATE is essentially the same in format as COPYSORT with the difference that files are copied according to their creation dates rather than alphabeticaly.

FILEDATE

FILEDATE is a flex utility used to change the FILE CREATION DATE in the disk directory. This is useful when the DATE bytes in FLEX have been inadvertently corrupted and the file dates are incorrect.

INFO & CINFO8X

INFO command is used to display the ATTRIBUTES of the FLEX disk directory. This command is especially useful prior to using the COPYSORT command to make a copy of a disk according to extensions as all the existing extensions are displayed in alphabetical order.

CINFO8X.CMD is used only with CIMIX FLEX, and displays the size of disk drive in use, as well as issuing an appropriate message if a non-existent drive is specified.

RELINK & RELINK82

RELINK is a flex utility used to relink all the available free sectors on a FLEX formatted disk, thus decreasing the number of drive head relocations on a disk which has become very fragmented from multiple file deletions. RELINK82 is for use on a South West Technical Systems 8209 or 8212 terminal only whereas RELINK is for general use.

SECTORS.CMD

The related SECTORS.CMD lists all the free sectors in the order of their linkage addresses and is useful to check the extent of the fragmentation of the free sectors, as well as finding the address of a faulty sector when a Flex error 9 or 10 is encountered.

RESQ

RESQ is a very useful utility which allows for the recovery of files that have been inadvertently or otherwise deleted.

XL

XL is a utility used to read text files. It loads the text file into memory and the user then can list the whole file, list lines or a single line, find a string of characters, etc.

It is similar to a line-oriented text editor but without the capability of making any changes to the file. It does however have two useful features In that the whole file or selected lines can be routed to the line printer or the disk by using the TSC 'P.CMD' or the 'O.CMD' in the command line. Thus it can be used to write out parts of a program such that only selected branches can be looked at, making interpretation of programs (BASIC or ASSEMBLY type) easier.

Machine-coded files can also be scanned for any text in them as all non-printable characters are replaced by a '.'.

Disk # 3
ASSEMBLER & DISASSEMBLER UTILITIES
$ 39.95

LINEFEED

This is a simple FLEX utility used to produce a text file suitable for study and modification after a disassembly using SUPERSLEUTH or any other disassembler.

The utility scans each line of text and when any of the mnemonics below are found, it places a following blank line. In this way all subroutines and interruptions to program flow (branches) are spaced out as discrete blocks of code, making scanning and modifications easier to do.

MATH

MATH is a flex utility used to work with upto 16 bit integers in binary, hexadecimal, decimal and octal. The program is useful in compiling lists of the number systems, for calculating offsets, addresses and masks.

SKIP

SKIP is a FLEX utility used to strip columns in a text file from colum 1 to 255 as the limits. This simple FLEX utility is especially useful to produce a text file suitable for editing or assembly with the assembler fields in column form. Often an assembly language source file is obtained or written with just single spaces between the fields.

Disk # 4
WORD - PROCESSOR SUPPORT UTILITIES
$ 49.95

FULLSTOP

FULLSTOP is a utility used with files, such as letters, reports, articles for publication, etc., produced by an editor or word processor. This utility checks whether the NEXT alphabetic character encountered in a file after a period (.), a question mark (?), an exclamation point (!), or a carriage return is in UPPER CASE. If it is not upper case then the user is prompted to enter the new character from the keyboard.

BSTYCIT.BAS (.BAC)

This together with the OS9 version BSTYLO_CIT, is an adjunct to STYLO for users with a dot-matrix printer, allowing the user to use STYLO's commands for bold-face, underline, overline, subscripts and superscripts without having to enter strings of printer codes. It prints out a Stylo generated text file just as if Stylo itself was doing the printing. At the moment they only support the C. Itoh dot-matrix printer (8510 & 1550) and the Epson FX80 (or MX80 with Graftrax).

NECPRINT

This printer filter for the Stlograph word processer converts NEC Spinwriter 5510/20 codes to C. Itoh 8510/5515 codes from STYLO. The source code is modular so that the revelant codes for other dot-matrix printers can be used.

Disk # 5
UTILITIES FOR INDEXING
$ 49.95

INDEXING PROGRAMS FOR TEXT FILES, REPORTS, BOOKS, ETC.

The suite of programs supplied help in the creation of a sorted index and a table of contents from text files prepared using an editor/word processor. As supplied, the programs are assumed to work with STYLOGRAPH files. If you are using a different W-P then edit the .BAS files INDEX, CONTENT, and PHRASES, where the line 'REM process STYLO command' appears, so that the commands reflect your W-P's syntax. Then 'compile' these customised versions. Stylograph uses a comma at the leftmost position to indicate a formatting command. Thus:

,pg is the command for a new page,
,pc char is the command for embedded printer codes, char being one character which must be printable (eg.: '@'),
,pn number specifies the starting number for physical page 1.
,* is a comment line, not printed at runtime.
,* ha number is a pseudo-Stylo command for the Table of Contents generator.

INDEX, Frequency and PHRASES all optionally display the unformatted text on the screen as it is processed, so that one can see how far the programs have progressed.

A Review QPL

QPL is a product of:
Compiler Products Unlimited
6712 E. Presidio
Scottsdale, Arizona 85254
Ph (602) 991 1675

Modern "high level languages" are "structured". Right? Well, not necessarily. A program written in QPL can be very much "unstructured". I can almost hear gasps from those who feel that "structured" programming is the only useable kind. However, there are times when some aspect of a given program could be improved by a departure from the rigid restraints imposed by some languages. QPL has it's roots in SNOBOL and features a subset of SNOBOL4 functions some of which have been enhanced. A primary objective in the design of QPL was to be easy on the programmer and I found that to be essentially so. After only one pass through the excellent manual I found it very easy to write simple programs to write data to disk and then retrieve and display it. The programmer is given very wide latitude in the design of his program solution and good programmers may still use it to develop orderly, maintainable software although this language does not force a particular "style" as does, for example, Pascal.

Also in contrast with Pascal, QPL is heavily oriented to "string" handling and manipulation. A point which brings us to the next surprise -- it is not only unnecessary to declare variables in advance, but they are not specified by "type". QPL doesn't use "data types" because variables are capable of handling text (alph-numeric data) or numbers interchangeably.

While it is necessary, of course, to pre-define arrays (which may be multi-dimensional and even have symbolically determined dimensions), but an initially defined (empty) array uses no space (except for an overhead of 3 bytes per element). Data is written to disk in the same compact format, forming a sequential text file that is considerably smaller than the same data in random access format, especially a data base that is "sparse", with many unused, null length variables. It is possible to design a program that has very large arrays such that, if filled, would completely overrun the available RAM. Presumably, the programmer would incorporate safeguards to protect a subsequent user from inadvertently crashing the system by entering that much data. Further, arrays may be used to hold other arrays, and these still others and so on. But my mind boggles in trying to envision anything beyond the third level of arrays.

QPL variables also provide for a capability called "NAME INDIRECTION" whereby the contents of a named variable may be used to retrieve the the contents of another by prefixing the first label with a "$". That is, assuming that the variable LABEL contains the data NAME and that the veriable NAME contains JOHN, then the line OUTPUT = $LABEL will cause the terminal to display JOHN. (While this concept is difficult to put into words, 6809 assembly language programmers will pick it up immediately as a result of their familiarity with "[LABEL]"). Actually, QPL's INDIRECTION feature may be extended to added levels by introducing more "$"s, as in $$$LABEL. However, the programmer must have made data links that allow this to function without error and it quickly becomes unwieldy.

These features (interchangeable data type variables, dynamic array sizes and string lengths) make QPL very well equipped to do many information storage and retrieval applications. Further enhancing this aspect is a comprehensive set of "Pattern Matching" tools. The basic concept is to construct a pattern (and alternates, if desired) in the format of the data match desired. The options available in this powerful tool are too varied to be described here, but those available should be adequate to satisfy any reasonable information retrieval requirement. In fact, this feature is one of the primary reasons for the existence of QPL. I would recommend careful study of this portion of the manual for those interested in writing any type of data management system as QPL provides the ability to easily perform some tasks that can only be done with difficulty (or not at all) in any other language available for use with Flex9 systems.

QPL treats numbers in an unusual way. As mentioned above, variables may contain numbers or alpha-numeric data interchangeably. As a result, the "dynamic string length" aspect of variables is also available to numbers, so a number may have as many digits as will fit into RAM. The number format used internally by QPL is proprietary, but input/output is in the form of base 10 floating point and may, optionally, be in exponential form with a "limit" of plus or minus 32000. Some limit! (No that's not a typo, it's "32 thousand"). If you like to impress your friends with big numbers, this is it! For amusement I tried subtracting 3 from 1E250. It took about 10 seconds, but gave me back that l-o-n-g string of 9's and a 7. Since the real world rarely produces or needs data in that degree of precision and since processing these long strings consumes a great deal of machine time, programmers would want to be alert to the need to restrict the number of significant digits to a realistic level. Fortunately, QPL provides a keyword "EXPO" to select outputs in exponential form and another, "APPROX", which may be used to truncate large numbers. However, APPROX has a quirk in that the number of significant figures returned is equal-to-or-greater-than that specified in the argument. That is, if 3 significant digits are specified, truncation may actually be to 5 digits. However a user defined function can be written to overcome this when a formatted output is required.

QPL is not going to be suitable to every programming job that comes along (is any HLL?). The unique ability to handle very large numbers is not accompanied by a strong math package. In fact, QPL has only the four arithmetic functions -- anything beyond that must be added in the form of user defined functions and are the responsibility of the programmer. If your assignment is to write an "industrial strength" number cruncher, an elaborate spread sheet, or a real time process controller, QPL is obviously not going to be your first choice for a compiler. Of course programmer written functions can be used to, in effect, add to the built-in functions of the language, including the addition of higher level math functions. QPL does not support a "library" function in the sense that pre-programmed modules can be inserted in the run stream at compile time, so these have to be added while editing the source code. Compiler Products Unlimited is in the process of building a user group library, so it would be wise to inquire whether the desired function is available and possibly avoid re-inventing a wheel.

Having mentioned programmer responsibility, I should go on to add that such great latitude in programming style is accompanied by added responsibility for the programmer. Aside from the self discipline to carefully organize and comment the program for subsequent maintenance, he must also be more than usually alert in protecting against user errors. For example, since there is no distinction between alpha-numeric and numeric variables, and since all variables are "global", the stage is set for a program that will "bomb" when the user enters an alpha into a variable that is later used an arithmetic operation. This will result in a non-recoverable error at run-time and the user will have to restart the run from scratch.

The source code for a QPL program is VERY compact when compared with most other compilers. But, since this is accomplished without resorting to the use of odd-ball symbols, it is also very readable. Well commented source is made easy by entering an asterisk (*) as the first character on a line, as with an

assembler. LABELS must begin with an alpha character in the left-most position on a line. All other statements are indented at least one space.

Programmers who are accustomed to writing a set of "procedures" and then tying these together with a "main program" will have to adjust their thinking a bit. QPL uses the "goto" as the primary method of controlling program flow. Five comparison functions are provided; EQ (equal), NE (not equal), LE (less than or equal), GE (greater than or equal and LT (less than). These comparisons are used with a "success or failure" flag to determine the action taken at a "goto" branch point. An example will make this clearer:

```
   COUNT = 0
NEXT_ONE OUTPUT = COUNT
   COUNT = COUNT + NE(COUNT,10)      :S(NEXT_ONE)
```

This forms a simple loop that will display on terminal the value of COUNT up to the value of 10. That is, as long as the NE condition Succeeds, the goto label NEXT_ONE prevails. Of course the sense of the comparison could be inverted by using EQ(COUNT,10) :F(NEXT_ONE), looping for as long as the equal test Fails. Other keywords (MATCH, IDENT) affect the global succeed/fail flag in a similar manner, and after the test the program will either goto the labelled code, or fall through to the next line.

Computer Products Unlimited refers to QPL as being a Very High Level Language, which means that the compiler output is an intermediate file that is later processed by one of several available "linkers". One function of the linker is to selectively add the required elements of the run-time interpreter, reducing the total size. Even so, the RAM requirement for a small program seemed astonishingly large because of the run-time code added by the linker. Larger programs will, of course, have a better ratio of source to object. But where RAM is a limiting factor, this could be a major consideration in selecting this language.

In summary, I found several things to like about QPL. It's easy to use, has excellent string handling features, and compiles quickly (assuming a fast terminal). On the other hand I would caution that its lack of math functions, slow processing speed, and large object code would thwart its use for some applications.

Art Weller
3217 Pagosa Ct.
El Paso, TX 79904

P.S. Did you know that 1/97 =
0.010309278350515463917525773195876288659793814432989690
721649484536082474226804123711340206185567?

Bit Bucket

Fred Stucklen
148 Winkler Rd.
E. Windsor, Ct. 06088

Dear Don:

First let me congradulate you on a fine magizine. Keep up the support for the 68XX families!

On the encloseed diskette is a program that I wrote while doing a control job that involved a matrix keypad. This routine had to be fast, short, and use as little external ram as necessary. The result was KEYSCAN.

This routine offers key debounce for both closure and release, as well as a repeat function like that found on most terminals. The CA2 line also will toggle high for a "BEEP" duration, allowing it to drive an external tone generator.

I hope you and your readers find this program useful.

```
 1.00* NAM Keypad Scan Routine
 2.00*
 3.00* WRITTEN BY:  F.W.Stucklen  March 14,1986
 4.00*
 5.00*
 6.00* The following is a general purpose matrix
 7.00* keypad scan routine. It was primarily written
 8.00* to be used within an interrupt routine, but
 9.00* may be called as a subroutine as long as it is
10.00* serviced at regular intervals. It has built in
11.00* debounce for key closure and release, as well
12.00* as a key repeat function if a key is held down.
13.00* This example was written for a 4X4 keypad
14.00* using one PIA 8 bit port. It can be easily
15.00* changed to an 8X8 matrix using two 8 bit PIA
16.00* ports. The following guidlines must be
17.00* followed for any changes:
18.00*
19.00*  1. ROWS are always configured on the PIA from
20.00*     the lowest bit upwards (ie, PA0 up).
21.00*  2. COLUMN are always configured on the PIA
22.00*     from the highest bit downwards (ie, PA7
23.00*     down).
24.00*  3. Bits in COLMASK are zeros for each active
25.00*     column.
26.00*  4. Bits in ROWMASK are ones for each active
27.00*     row.
28.00*  5. SCNRAT sets up the key closure and release
29.00*     debounce times. Actual time is SCANRATE
30.00*     times COLMAX, where SCANRATE is the rate
31.00*     at which SCAN is called.
32.00*  6. RPTMAX is the initial delay for the repeat
33.00*     fuction, and is equal to the number of
34.00*     SCAN's before a repeat fuction is started.
35.00*  7. RPTMIN sets the repeat rate.
36.00*
37.00* SCAN uses seven bytes of RAM that can be
38.00* anywhere as long as they are in the same page
39.00* of memory.
40.00* This program was written with Windrush Micro
41.00* Systems MACE (seven letter labels were used).
42.00* Lines starting with "*" include PL9 procedure
43.00* names, allowing SCAN to be put into GEN
44.00* statements.
45.00*
46.00*
47.00*RAM EQU $0F98 SCAN RAM usage area....
48.00*COLPORT EQU $F720 COLUMN I/O PORT ADDRESS
49.00*ROWPORT EQU $F720 ROW I/O PORT ADDRES
50.00*
51.00*
52.00*ROWMAX EQU 4 # OF KEYBOARD ROWS
53.00*COLMAX EQU 4 # OF KEYBOARD COLUMNS
54.00*COLMASK EQU $F0 Column mask (AND'ed)
55.00*ROWMASK EQU $0F Row mask (OR'ed)
56.00*MATVAL EQU $0F Max key value
57.00*SCNMAX EQU 2 # scans for debounce
58.00*RPTMAX EQU $20 Initial repeat delay
59.00*RPTMIN EQU $10 Keypad repeat rate
60.00*BEEPTIM EQU $10 Beep timer duration
61.00*
62.00* CPU RAM
63.00*
64.00*KEY RMB 1 Keypad key buffer
65.00*KYSTAT RMB 1 keypad status
66.00*LSTKEY RMB 1 Last key found
67.00*KEYTMP RMB 1 Temp lastkey
68.00*COLCNT RMB 1 Last column tested
69.00*SCNCNT RMB 1 Last row tested
70.00*RPTCNT RMB 1 Keypad repeat timer
71.00*BEEPCNT RMB 1 BEEP counter
72.00*
```

```
73.00= ORG $9000 FOR REFERENCE ONLY...
74.00=
75.00=*asmproc keyscan;
76.00=SCAN PSHS DP,A,B,I
77.00= LDD #KEY POINT AT TEMPS AREA
78.00= TFR A,DP SET DP TO THAT AREA
79.00= LDA <KYSTAT LAST KEY READ?
80.00= LBMI BEEP NO, SO RETURN
81.00= LDB COLPORT SET PROPER COLUMN LOW
82.00= ORB #COLMASK
83.00= STB COLPORT SET'EM HIGH FIRST
84.00= LDA <COLCNT CALC KEY TABLE ADR
85.00= LEAI >KEYTAB,PCR POINT AT THE KEY TABLE
86.00= INC A
87.00=SCAN3A DEC A
88.00= BEQ SCAN3B
89.00= LEAI #ROWMAX,I
90.00= BRA SCAN3A
91.00=SCAN3B LDA <COLCNT CURRENT COLUMN COUNTER
92.00= INC A ADJUST IT
93.00= LDB #$EF
94.00=SCAN4 DECA
95.00= BEQ SCAN5
96.00= SEC
97.00= ROL B ROTATE THE MASK BIT
98.00= BRA SCAN4 TO THAT COLUMN.
99.00=SCAN5 ANDB COLPORT SET THAT COLUMN BIT LOW
100.00= STB COLPORT
101.00= LDA ROWPORT READ THE KEYBOARD
102.00= ORA #ROWMASK MASK UNUSED BITS
103.00= STA <KEYTMP
104.00= CLR A TABLE OFFSET POINTER
105.00= LDB #$01 LOOK FOR A VALID INPUT
106.00=SCAN6 PSHS B SAVE THE MASK
107.00= ANDB <KEYTMP BREG IS THE ROW MASK
108.00= BEQ SCAN6A LOW=VALID INPUT
109.00= PULS B RESTORE THE MASK
110.00= ASLB SLIDE THE BIT OVER ONE...
111.00= INC A NEXT ROW
112.00= CMPA #ROWMAX LAST ONE??
113.00= BNE SCAN6
114.00= LDA <COLCNT NO MATCH THIS COLUMN
115.00= INC A
116.00= CMPA #COLMAX LAST COLUMN
117.00= BSE SCAN7 YES, SO SERVICE IT
118.00= BRA SCAN15 INC COLUMN COUNTER
119.00=SCAN6A PULS B MATCH: RESTORE STACK.
120.00= BRA SCAN10
121.00=*
122.00=* NO KEY PRESSED ANY COLUMN
123.00=*
124.00=SCAN7 DEC <SCNCNT DEC ONE SCAN COUNTER
125.00= BPL SCAN14 NOT ZERO, SO RETURN
126.00= LDA #SCNMAX RE-INIT IT
127.00= STA <SCNCNT
128.00=SCAN8 LDB #$FF INVALID KEY PRESSED
129.00= LDA <LSTKEY LAST KEY SAME AS KEY???
130.00= CMPA <KEY
131.00= BNE SCAN9
132.00= STB <LSTKEY
133.00= BRA SCAN14 AND EXIT
134.00=SCAN9 STB <KEY SET KEY=$FF
135.00= BRA SCAN14
136.00=*
137.00=* VALID KEY SO CHECK FOR TABLE MATCH...
138.00=* AND LAST KEY READ....
139.00=*
140.00=SCAN10 LDB A,I GET TABLE DATA
141.00= CMPB #MAXVAL VALID CLOSURE?
142.00= BGT SCAN8
143.00= LDA #SCNMAX RE-INIT SCAN COUNTER
144.00= STA <SCNCNT
145.00= CMPB <LSTKEY SAME AS LAST KEY?
146.00= BEQ SCAN10
147.00= STB <LSTKEY NO: SAVE FOR DEBOUNCE
148.00= BRA SCAN14 AND EXIT
149.00=SCAN11 TST <KYSTAT
150.00= BEQ SCAN12 LAST KEY BEEN READ?
151.00= BRA SCAN14 NO, SO EXIT
152.00=SCAN12 LDA <LSTKEY LSTKEY=KEY?
153.00= CMPA <KEY
154.00= BEQ SCAN12A SAME: REPEAT TIME??
155.00= LDB #RPTMAX 1'st TIME: MAX RPT TIME
156.00= BRA SCAN12B
157.00=SCAN12A DEC <RPTCNT
158.00= BNE SCAN14 @=RPT TIME
159.00= LDB #RPTMIN SET TO MIN RPT TIME
160.00=SCAN12B STB <RPTCNT
161.00=*
162.00=* REPORT VALID KEY
163.00=*
164.00=SCAN13 STA <KEY SAVE IT
165.00= INC <KYSTAT
166.00= LDA COLPORT+1
167.00= ORA #$08 SET CA2 HIGH
168.00= STA COLPORT+1
169.00= LDA #BEEPTIM INIT BEEP TIMER
170.00= STA <BEEPCNT
171.00=SCAN14 LDA <COLCNT UPDATE COLUMN COUNTER
172.00= INCA EACH TIME THRU
173.00= CMPA #COLMAX LAST ONE ??
174.00= BLT SCAN15
175.00= CLRA
176.00=SCAN15 STA <COLCNT
177.00=BEEP TST <BEEPCNT BEEPER OFF?
178.00= BEQ SCANDON ALREADY OFF
179.00= DEC <BEEPCNT ADJUST IT
180.00= BNE SCANDON NOT TIMED OUT YET
181.00= LDA COLPORT+1
182.00= ANDA #$F7 TURN OFF BEEP
183.00= STA COLPORT+1
184.00=SCANDON PULS DP,A,B,X,PC
185.00=*
186.00=* THIS IS THE VALID KEY TABLE.
187.00=* IT IS GROUPED IN FOURS FOR EACH
188.00=* OF THE COLUMNS AND ROWS.
189.00=*
190.00=*                ROWS
191.00=*              ----------
192.00=*
193.00=KEYTAB FCB $1,$2,$3,$A  !
194.00= FCB $4,$5,$6,$B  !  <-- COLUMNS
195.00= FCB $7,$8,$9,$C  !
196.00= FCB $F,$0,$E,$D  !
197.00=*
198.00=*/*****************************/
199.00=*/*   HARDWARE INIT           */
200.00=*/*****************************/
201.00=
202.00=* The keypad I/O port is initialized to
203.00=* support the row and column formates you need.
204.00=* The CA2 line of the PIA is used to enable
205.00=* an external tone generator. This routine
206.00=* assumes the use of PA0-PA7 of a PIA.
207.00=* for a 4X4 keypad matrix.
208.00=*
209.00=*procedure key_init : byte cnt;
210.00=HARDINIT LDX #COLPORT
211.00= CLR 1,X
212.00= LDA #COLMASK
213.00= ANDA #ROWMASK
214.00= STA 0,X
215.00= LDA #$36 DDR BIT HIGH, CA2 OUPUT
216.00= STA 1,X LOW.
217.00= LDX #KEY-1 CLEAR THE TEMPS
218.00=INIT0 INI
219.00= CLR 0,X
220.00= CPX #BEEPCNT
221.00= BNE INIT0
222.00= RTS
223.00=
224.00=* This is an example of how to read the keypad.
225.00=* KYSTAT is always checked, and cleared ONLY
226.00=* after the value of KEY has been read. No
227.00=* SCANS are performed if the last key has not
228.00=* been read (time saver?).
229.00=
230.00=GETKEY LDA KYSTAT HAS A KEY BEEN PRESSED?
231.00= BEQ GETKEY NO, SO WAIT
232.00= LDA KEY YES, SO READ THE KEY
233.00= CLR KYSTAT
234.00= RTS
235.00=
236.00=
```

CERTIFIED SOFTWARE CORPORATION

616 CAMINO CABALLO, NIPOMO, CA 93444 USA TELEPHONE: 805-929-1395 TELEX: 467013

Dear Don,

I would to tell you how pleased I am with the Mustang-020 system I received last week. At the very minimum it runs my software 3 times faster than my VME/10 system (by the way, I can't recommend the VME/10 system to anyone, or the Microware OS9 port for it).

Taking advantage of the RAM DISK and ebility to leave all the programs I normally need in memory, speeds development even further. The only criticism I have of the system is that the default aerial port speed is in EPROM, you might want to consider some other way to select between 9600 and 19200 baud.

Sorry I can't buy more of them, but I intend to recommend the system to my 6809 customers looking to upgrade to the 680xx family.

Sincerely,

Robert A. Reimiller

Robert Reimiller
President, Certified Software Corporation

Editor's Note: Thanks Bob for the nice comments. We are hearing nothing but raves from new users of the Data-Comp MUSTANG-020!

We think that our OS-9 port is one of, if not the best port of OS-9, on any 68XXX system we have seen.

As to the RS-232 port speed being in EPROM. That is a slight inconvience that I hope will be corrected in the future. We can custom set the baud, prior to shipping, at a nominal cost. However, most all (all except one of all the MUSTANG-020 systems we have shipped) are using the 19.2Kb terminal speed.

Thanks again Bob, and your kind "word of mouth" recommendation is working. We have already seen the results. We get calls all day long from potential users who marvel at the quality, price and power of the MUSTANG-020. Some have been from folks getting your fine newsletter.

DMW

\- - -

Micro 68 Journal
5900 Cassandra Smith
Box 849, Hixson, Tenn.
USA 37343

Gentlemen:

I would like to express my appreciation to 68' Micro, the staff, the contributing editors and the many readers that add so much to the magazine. I guess I would be one of those "old hackers" that Don Williamms mentions in replying to Mickey Ferguson in the February issue. I think the first I heard of a "computer on a chip" was mentioned in, I believe, Popular Electronics. I wrote away and got Scelbi Publication's manual on machine language programming for the 8008. Some time later American Microsystems brought out a single board computer called the EVK 99. That was the first piece of hardware I got. I expanded that up to 16k of memory and learned programming on it. I got Scelbi's fine 6800 cookbook and wore the book out. Those were the days I wrote assembly code on paper, assembled the code by hand and typed in the hex code into the machine. Those were the days when I had to calculate the code for the branch instructions for relative addressing. That was when I learned to count in hexadecimal backwards. I think it must have imprinted on me because I still think of software projects in terms of sssembly language code. After the EVK 99 I upgraded to a SWTP box, added a Teletype machine for a printing terminal, then varioua "glass teletypes". Still later SSB's disk controller board, the DC8-4A, 2 eight inch drives and SSB's disk operating system DOS69D. Along the way I picked up boards from Percom, Data Systems 68, Acorn, Peripheral Technology, Digital Research, Thomas, Unique Technology, Robertson and made up a few boards myself. Those served me for several years but software for DOS69 is scarce and nothing new was appearing for DOS69D so I have recently implemented Star-dos on my system. Peter Stark is to be complemented on his fine job on this, especially his latest version with all the upgrades.

As soon as I got Star-dos configued to run under SSB's DCB-4A I ordered PL/9 and am in the process of wearing out the manuals thumbing back and forth as I write code. An old dog can learn new tricks, it just takes a little longer than it used to. I haven't written anything big yet in PL/9, just utility type programs that occupy about a page or so of source code. This one that I am including is a program to do a limited batch file processing, similar to "EXEC" in DOS69D. The idea is that you can put multiple commands in a text file and executing them in a batch. The command syntax is "EXECUTE MYFILE" where "MYFILE" defaults to "MYFILE.TXT". "MYFILE" would contain one or more lines of commands each line 127 or less characters, for example:

```
LIST xxxx
ASMB xxxx,yyyy
yyyy
```

I haven't tried to use "MEMEND" to fit the program exactly at top of free memory but you can change the ORIGIN statement to fit your particular machine. You would also have to change the location of "FILENAME_BUFFER" so the buffer dos not overlay the program. I have specified a size of 1000 bytes for FILENAME_BUFFER but if you anticipate large batch files you can change the size of the buffer. The line INCLUDE STAR_DOS is really calling "FLEX.LIB". I just changed FLEX to STAR_DOS on my files. I see Peter Stark has included a couple of programs on his latest release of STAR-DOS called INPIPE and OUTPIPE. I haven't checked them out yet but they might be doing something similar to "EXECUTE".

Yours truly

Walter Isaacson
19 1614-22 Ave., S.W.
Calgary, Alberta
T2T 0R8

```
/* PROGRAM TO EXECUTE COMMANDS FROM A TEXT FILE */
/* by Walter Isaacson                           */
/* 19 1614-22 AVE., S.W.                        */
/* CALGARY, ALBERTA                             */
/* T2T 0R8                                      */

CONSTANT CR = $0D;

AT $9000: BYTE FILENAME_BUFFER(1000);
/* CHANGE THIS IF YOUR COMMAND FILES ARE TOO LARGE */
/* IF YOU MAKE THE BUFFER LARGER DONT FORGET TO MOVE IT DOWN */
/* IN MEMORY SO IT DOSN'T OVERLAY AND WIPE OUT THE PROGRAM. */
AT $C080: BYTE LINE_BUFFER(128);
AT $CC14: INTEGER LINE_POINTER;
AT $C840: BYTE FCB, ERROR(319);


ORIGIN = $9400;

INCLUDE STAR_DOS;  /* really FLEX.LIB */

PROCEDURE OPEN_FILE;
   GET_FILENAME(.FCB);
      IF FCB(1) THEN BEGIN
         REPORT_ERROR(.FCB);
         STAR_DOS;
      END;
   SET_EXTENSION(.FCB,1);
      IF FCB(1) THEN REPORT_ERROR(.FCB);
   OPEN_FOR_READ(.FCB);
      IF FCB(1) THEN BEGIN
         REPORT_ERROR(.FCB);
         STAR_DOS;
      END;
ENDPROC;

PROCEDURE READ_FILE( BYTE .FILENAME_BUFFER): BYTE CHAR;
   REPEAT
      CHAR = READ(.FCB);
         IF FCB(1) = $8 THEN BREAK; /* THIS IS THE CHECK FOR END OF FILE */
         ELSE BEGIN
         IF FCB(1) THEN BEGIN
            REPORT_ERROR(.FCB);
            BREAK;
            END;
         END;
         FILENAME_BUFFER = CHAR;
         .FILENAME_BUFFER = .FILENAME_BUFFER + 1;
```

```
      FILENAME_BUFFER = $00;
    /* I AM USING "NULL" TO MARK THE END OF THE STRING IN THE BUFFER */
    FOREVER;
  CLOSE_FILE(.FCB);
  IF FCB(1) THEN BEGIN
    REPORT_ERROR(.FCB);
    STAR_DOS;
  END;
ENDPROC;

PROCEDURE EXECUTE_COMMAND(BYTE .FILENAME_BUFFER, .LINE_BUFFER);
  BYTE .POINTER;
  REPEAT
    .POINTER = .LINE_BUFFER;
    LINE_POINTER = .LINE_BUFFER;
    IF FILENAME_BUFFER = $00 THEN STAR_DOS;
      REPEAT
        POINTER = FILENAME_BUFFER;
        .FILENAME_BUFFER = .FILENAME_BUFFER + 1;
        .POINTER = .POINTER + 1;
        POINTER = CR;
      UNTIL FILENAME_BUFFER = CR;
      .FILENAME_BUFFER = .FILENAME_BUFFER + 1;
      CALL $CDAB; /* THIS IS STAR-DOS EXECSO ENTRY POINT */
  FOREVER;
ENDPROC;

PROCEDURE MAIN;
  OPEN_FILE;
  READ_FILE(.FILENAME_BUFFER);
  EXECUTE_COMMAND(.FILENAME_BUFFER, .LINE_BUFFER);
```

MICRONICS RESEARCH CORP.
Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA.
CANADA. V2S 1E2

Dear Don,

This time around I'll try to concentrate a little more on XBASIC, and forget about BEDIT until I get some response regarding original authorship. So let's begin with a simple Decimal/HEX conversion routine, which I recently sent to a new correspondent from Southern Africa.

```
10  H$="": INPUT D
20 REM CONVERT DECIMAL INTEGER TO HEX STRING
30  E=INT(D/16): F=D-E*16: IF F>9 THEN F=F+7
40  H$=CHR$(F+48)+H$: IF E<>0 THEN D=E: GOTO 30
50  PRINT H$: GOTO 10
```

Let's talk about this little program for a while, and see where it leads us. Line 10 sets the HEX-string to a NUL, and requests input of a Decimal number - any length. The actual conversion routine is composed of Lines 30 and 40; Line 30 divides the decimal number by 16, retaining the quotient in 'E' and the remainder in 'F' (as a HEX remainder 0 - 15). To prepare the remainders 10 - 15 for conversion to the letters A - F we add 7 to their decimal value. Then Line 40 converts the remainder to a CHR$ and prefaces this string to the previous value of H$, returning for further computation if the conversion is not yet complete, ie if E has not been eliminated by the current operation. Finally the result is displayed, and back again for a fresh input.

Not a particularly remarkable program, though quite effective for its desired purpose, and there doesn't seem to be much room for improvement in its short 2-line conversion. Before we continue, however, note the useful technique of indenting REMs by only one space, and the rest of the program by 2 spaces. This makes it easy to locate all the REMs in your programs.

First of all, observe that Line 40 is forced upon us because we must convert the remainders 10 - 15 into their corresponding letters A - F, and yet preface H$ with a new CHR$ whether or not such conversion has occurred. There would appear to be no way we can comfortably tack Line 40 onto the end of Line 30 and still make the program work. I hope those already in the know will bear with me while I discourse on the fact that the whole of the IF-THEN statement in Line 30 can be compressed into a compact logical function **and included in the CHR$ function.** Thus :

```
30  E=INT(D/16): F=D-E*16: H$=CHR$(F+48-7*(F>9)): IF
    E<>0 THEN D=E: GOTO 30
40  deleted
```

Let's examine the extra enclosure -7*(F>9) in a little more detail. The part (F>9) makes use of XBASIC's implementation of Boolean logic, whereby the result is -1 if the statement is TRUE and 0 if it's FALSE. So, (F>9) will be replaced by 0 or -1, depending on whether it's FALSE or TRUE, and our original F+48 will correspondingly have either (0*7) or (-1*7) subtracted from it. Of course, subtracting -7 is the same as adding +7. We'll elaborate a bit more further on in my letter. For now, let's consider that if input of D is restricted so as not to exceed 32767, we can both shorten and speed up the program by replacing 'D' with 'D%', and so on. Further, this would allow E=INT(D/16) to be shortened to E%=D%/16, which could lead me off on yet another tangent, but I'll resist the temptation for the moment, and stick with my current subject.

Here's another situation where we can make use of logic functions. Suppose we had the trivial situation where a message had to be TABbed to say, 30, if the response to a question were "Y" (for YES) or to 40 if the reponse were "N". For example :

```
10 INPUT "Do you like this (Y or N) ",Q$
20 IF Q$="Y" THEN PRINT TAB(30); ELSE PRINT TAB(40);
30 PRINT "Good!"
```

In this case, Lines 20 and 30 can be combined so :

```
20 PRINT TAB(30-10*(Q$="N")); "Good!"
```

Note that this will give a minimum TAB of 30, but bump it by 10 if the response is "N" (ie -10*-1 = +10). A "Y" response would evaluate as FALSE, giving -10*0, or 0 change to the TAB of 30. And what a saving in program length!!!

To close, let's imagine the ridiculous situation where say X has to be divided by 10 if Y<=6 otherwise divided by 20, and also multiplied by 11 if Z<=9 otherwise multiplied by 15. Compare this :

```
10 IF Y<=6 THEN X=X/10 ELSE X=X/20
20 IF Z<=9 THEN X=X*11 ELSE X=X*15
```

with this :

```
10 X = X / (10-10*(Y>6)) * (11-4*(Z>9))
```

With a little practice, it soon becomes quite natural to read (10-10*(Y>6)) as 'divide by 10, or by 20 (ie 10+10) if Y>6', and similarly with (11-4*(Z>9)) as 'multiply by 11 or by 15 (ie 11+4) if Z>9'.

These are but a few typical examples, which should serve not only as a guide to more complex functions, but perhaps help clear up the interpretation of any such cases you may already have come across.

In closing, I would like to thank those readers who have taken time out to give me encouragement to carry on. Writing isn't really my "thing", but I do enjoy teaching (that was a boyhood dream which never materialised), so to those of you who suggested that I write a regular column, I can only say that I don't think I could keep it up on a regular basis. However, I will continue to send in my letters as long as I feel I have something worthwhile to pass on, and, of course, as long as both Don and you folks can put up with me. See you next time on the subject of INT(xx).

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Sincerely,

Bob

R. Jones
President

Mickey E. Ferguson
P.O. Box 87
Kingston Springs
Tennessee 37082

Mr. Donald M. Williams
Publisher 68 Micro Journal
5900 Cassandra Smith Rd.
Hixson
Tennessee 37343

Dear Sir:

I am writing this in the hope that some of your readers might be able to assist a young friend of mine in India. His name is P. Meganathan and he is a collage student in what used to be called Madras India. Mega's father is deceased and his mother is a school teacher, so they have a rather small income. Mega is starving for information about computers and this information is unavailable to him. His is studying mechanical engineering and his family does not have the funds necessary for him to take any computer courses. There are very few books available to him on computers and he cannot afford them anyway.

I would like to ask your readers if they have any old books and/or magazines on computers to please send them to Mega at the following address:

P. Maganathan
6, Ponnagaram 4th Street
Madurai - 10 Madurai District
Tamil Nadu State South India
India Pin 625010

I hope that you will print this letter and I hope that your readers will help Mega and his friends quench their thirst for the knowledge we have so freely available to us here in the good old U.S. of A.

I am also looking for a way to purchase an inexpensive computer system for delivery to Mega. So that he can actually get his hands on a real computer. I would just get a C-64 or a CoCo and ship it to him. But anything I can buy would be to U.S. standards and thus unusable in India. If anyone can help me in this, please write to me!

Sincerely,

Mickey E. Ferguson

John J. Fiorino
518-65th Street
Brooklyn, N.Y. 11209

I would like to get some information if anyone has the wiring of a OKIDATA or STAR printer to SWTPC aerial & parallel ports.
I know that you have received this request before, and you have said that there is no new writting about the 6800 system, well how about reprinting some of the old writting for the new members which startup with used systems.
I'd also like to know what SWTPC is up to lately.
Keep up the good work.

John J. Fiorino

1019 Weatherdon Ave.
Winnipeg, Manitoba
Canada R3M 2B5
10 Feb 86

68' Micro Journal
5900 Cassandra Smith
P O Box 794
Hixson, Tn 37343
USA

Dear Sirs:

Please send me the two-disk 5-inch set of reader service disks containing Flex-09 Kermit, as mentioned in the February '86 issue of 68' Micro Journal. I am enclosing a US money order for $19.95.

I have been looking for a Kermit for my Flex system for some time now, and was pleased to see the article in your magazine. However, the article consisted mostly of a command description, and I don't think it really showed readers how useful Kermit can be. I am enclosing a flyer from Columbia University that you may find interesting. We use Kermit at work to transfer files between Amdahl mainframe computers and MS-DOS micros, and I am looking forwared to doing the same with my 6809 at home.

Yours Truly

J. Gary Mills

Columbia University Center for Computing Activities

THE KERMIT FILE TRANSFER PROTOCOL

June 1985

Kermit is a protocol for transferring sequential files between computers of all sizes over ordinary asynchronous telecommunication lines using packets, checksums, and retransmission to promote data integrity. Kermit is non-proprietary, thoroughly documented, and in wide use. The protocol and the original implementations were developed at Columbia University and have been shared with many other institutions, some of which have made significant contributions of their own. Kermit is presently available for more than 400 different machines and operating systems, and additional versions are always under development. Current implementations include:

Unix (V7, 4.x BSD, System III, System V, Xenix, Venix, PC/IX; C language)
Software Tools (various systems; Ratfor)

Burroughs B6800, B7900 (Algol)
Cray-1, Cray-XMP (CTSS; Fortran-77)
CDC Cyber 170 (NOS, NOS/BE; Fortran-77)
Data General Nova (RDOS; Fortran-5)
Data General AOS (Fortran-5), AOS/VS (Pascal)
DEC PDP-11 (RT11,RSX11M(+),RSTS,P/OS,TSX+; Macro-11), (MUMPS; MUMPS-11)
DEC VAX-11 (VMS; Bliss-32 or Macro-32), (VMS; Pascal/Fortran)
DECsystem-10 (TOPS-10; Bliss-36, Macro-10)
DECSYSTEM-20 (TOPS-20; Macro-20)
Harris 800 (VOS; Pascal)
Honeywell (MULTICS; PL/I), DPS-6.6 (GCOS; C, B), CP6 (Pascal)
Hewlett-Packard 1000 (RTE-6/VM; Fortran), HP3000 (MPE; SPL or Fortran)
IBM 370-Series (VM/CMS, MVS/TSO, MVS/GUTS, MTS, MUSIC; Assembler)
Perkin-Elmer 3200 Series (OS32; Fortran)
PRIME (PRIMOS; PL/P)
Sperry/Univac-1100 (EXEC; Assembler or Ratfor or Pascal)
Tandem (Monstop; TAL)

CP/M-80 (about 20 different systems; ASM)
CP/M-86 (DEC Rainbow, NEC APC, and several other systems; ASM86)
MS-DOS, PC-DOS (IBM PC,XT,AT, DEC Rainbow, and many other systems; MASM)
UCSD p-System (IBM PC, Terak, and other systems; Pascal)

Alpha Micro 68000 (Alpha 68K Assembler)
Apollo (Aegis; Pascal)
Apple II 6502 (Apple DOS; DEC-10/20 CROSS or Apple Assembler)
Apple Macintosh (Supra CC C)
Atari (DOSs; Action!)
Commodore 64 (DEC-10/20 CROSS or FORTH)
DEC Pro-300 Series (P/OS; Bliss-16 or Macro-11), (Pro/RT; Macro), (Venix; C)
Intel Development System (ISIS; PL/M)
NCR Tower (OS 1.02; C)
Perq (Pascal)
TRS80 Models I,III,4 (TRSDOS; ASM), Model 16 (Xenix; C), Color Computer (Asm)

The IBM mainframe Kermits work only with asynchronous TTY connections through 3705 or equivalent front ends. The VM/CMS and MVS/TSO versions also have an

option to allow file transfer through Series/1 or other front ends supporting the Yale ASCII Communications System; beyond that exception, Kermit cannot transfer files in the IBM synchronous 3270-style full screen terminal environment.

The Kermit software -- including source -- is furnished free, without license, and with no restriction on copying or redistribution except that it should not be sold for profit, and that any copyright notices must be left intact. Under certain conditions (described in a separate document) software producers may include Kermit protocol in their products. Kermit software and documentation is furnished without warranty of any kind, and neither Columbia University, nor the individual authors, nor any institution that has contributed Kermit material, acknowledge any liability for any claims arising from the use of Kermit.

Although the Kermit software is free and unlicensed, Columbia University cannot afford to distribute it for free because the demand is too great. To defray our costs for media, printing, postage, labor, and computing resources, we require a moderate distribution fee from those who request Kermit directly from us. The schedule is given on the Kermit Order Form. Alternate sources for Kermit material are listed below.

Kermit is distributed by Columbia University only on 9-track magnetic tape, suitable for reading on most mainframe and minicomputers. It is assumed that Kermit will be ordered in this form by institutional computer centers, whose professional staff will take the responsibility for ``bootstrapping'' the microcomputer versions from the tape to diskettes for their users.

Documentation includes the Kermit User Guide, which contains complete instructions for installing and using the major implementations of Kermit, the Kermit Protocol Manual, which is a guide for writing a new implementation of Kermit, and the manuscript from the Kermit article that appeared in the June and July 1984 issues of BYTE Magazine.

Once you receive Kermit, you may redistribute it on your own terms, and are encouraged to do so, with the following stipulations: Kermit should not be sold for profit; credit should be given where it is due; and new material should be sent back to Columbia University so that we can maintain a definitive and comprehensive set of Kermit implementations for further distribution.

ALTERNATE SOURCES:

Kermit is also available to users of the BITNET network via a server at host CUVMA (BITNET users type ``SMSG RSCS MSG CUVMA KERMSRV HELP'' for further information); the Internet (via anonymous FTP from host CU20B, in the area KERMIT); UUCP from host oh5etc; and on magnetic tape from user groups like DECUS and SHARE. IBM PC-format MS-DOS Kermit floppies can be ordered from PC-SIG, Santa Clara CA, 14081 730-9291.

Dear Don,

After receiving the March issue of 68 Micro I cannot hold my anger any longer. This has to do with the continued use of GOTO's, in particular in the March "C" User Notes, and in past months in other columns, such as the Cobol notes of the August through September 85 issues.

Much blood and tears have been shed on this issue, and I thought that by now a journal such as yours, devoted to an excellent service to the 68 family, would not give us rat bane.

I am not knocking the columns, because they are badly needed. But a language such as C deserves to be treated better. It is just a mentality that some people have, they can't do away with the offensive GOTO, when in reality it takes a different tack, and with a different strategy than that displayed in the "C" User Notes, a C program need not employ a single GOTO. I am sick and tired of being fed spaghetti. Please have the authors recast their programs without them, otherwise I sugest that they take my C programming course (from the Northern Virginia Community College). For years I have taught Cobol without ever using this ignominious construct. If you can't think Structured Programming, and cannot cast a program without using spaghetti, then don't give us the programs. C is by definition a structured language, one that lends itself beautifully to structured programming, but oh! no, somebody has to spoil it.

Enough is enough.

Sincerely,

Albert Pinto
3303 Horseman Lane
Falls Church, Va 22042

John Moorfoot
2 Yarrayne St.
Leopold, VIC 3224
Australia

Attn: Don Williams Sr.

Dear Don,

I am enclosing a 5 1/4" disk with MODEM68 version 1.0.4. I have made a number of changes since version 1.0 which you published in 1984. These include new modules to support 6551 ACIA's, an XMODEM mode to allow control of the computer running MODEM68 from a terminal remotely connnected via the modem port, a help facility, and patches to existing modules to support cyclic redundancy checking on file transfers. I have also included a switch to enable two FLEX computers to transfer binary files correctly. The READ.ME file details descriptions of the various modulea, differences between versions, and hints on adapting to different environments.

The differences between this and the previous version are quite extensive in some modulea, so I have not attempted to just give you a list of patchea. I suggest that users completely replace their existing version 1.0 where possible. Users in Australia can obtain version 1.0.4 from TARDIS RCPM. I would also like to thank everyone who has helped me with bug reports, and expecially Howard Greenhill for his assistance with testing new versions.

Yours sincerely

John Moorfoot

1810 N.E. Fremont
Portland, OR 97212
Phone (503) 284-2831
March 12, 1986

Dear Don,

Last year I bought a 9511A math co-processor for my G1M1X 05+ (6809) board and am having trouble writing software for it. Would one of your readers help? (For a fee?) Thank you.

Yours truly,

Gary Lemoine

Don Williams
'68 Micro Journal
5900 Cassandra Smith Road
Hixson, TN 37343

4490 Yukon Court #2A
Wheatridge, CO 80033
February 26, 1986

Dear Don:

I've got some comments about the March issue of '68 Micro and thought you would like to hear them.

First, I think I saw a couple of errors:
The sample assembly code on Page 8:

```
 CLRA
 CLRB
LOOP STD ,--X
 BNE LOOP
```

will not loop because the condition flags are set according to the value in the register being stored (D) not the address register (X).

Second, I can't be absolutely sure without trying the program on the 68000, but the list program (on page 22) may terminate if it hits a $ff byte in the file (I know that probably won't show up in a text file).

The reason is that variable c is declared char. Before it is compared to EOF (-1) it will be sign-extended from char to int (at least, according to Kernighan and Ritchie and Microware C on the 6809). If c equals 0xff (255) it will be sign-extended to (int) 0xffffffff (-1 or EOF). The solution is to declare c as int. This may actually speed up the program a bit as it will not have to do a sign extension before comparing c to -1.

There is another possible reason for AT&T's reluctance to let Microware implement OS-9 on the UNIX PC (OS-9 User Notes, page 9). AT&T paid for the development of UNIX and profits from the sale of UNIX licenses. It could be that they do not want another operating system with superior performance and more efficient memory usage to "show up" the UNIX system.

Re language efficiency: (page ??)

Maybe pL-9 needs the logical and operator (&&) from the C language. The statement (in C):

```
if(a == 2 && b == 1)
```

is equivalent to

```
if(a == 2)
    if(b == 1)
```

You can even make a header file (I call mine logic.h) that defines EQUALS as == and LAND (for Logical AND) as && so that the first line can be written as:

```
if(a EQUALS 2 LAND b EQUALS 1)
```

This has the added benefit of preventing the mistake of writing the line as:

```
if(a = 2 && b = 1)
```

which tells the program:

```
make a equal to 2
make b equal to 1
execute the conditional code because
a and b are non-zero
```

By the way, you can shorten ADTEST (page 8) with the following code:

```
value fdb 533   modeled after original code
*               in 68' Micro for FLEX
temp rmb 2      don't use PCR or extended storage
*               in OS-9!
start ldd value
 aslb
 rola
 aslb
 rola
 addd value
 aslb
 rola
 std temp
 rts
 end start
```

****

I thought your readers might want to be aware of a couple of bugs in Microware C (on CoCo OS-9, at least).

First, the following code may not work properly:

```
unsigned value; /* only happens with unsigned
variables */
    ... previous code ...
    if(value>0)
        do something;
```

A comparison of an unsigned value to zero compiles to the following code:

```
    ldd ,s   assuming ,s is the location of value
    cmpd #0
    bls nope skip past code if <=0
    whatever
nope
```

This <u>will</u> work until the code is run through the optimizer. The optimizer will remove the "cmpd #0" leaving:

```
    ldd ,s   assuming ,s is the location of value
    bls nope skip past code if <=0
    whatever
nope
```

Note that "bls" will branch if the carry flag is set, but "ldd, s" does not affect the carry flag. So the program will branch depending on the last instruction that affected the carry flag.

You can program around this by changing the comparison to:

```
if(value != 0) /* not equal to zero */
```

This will depend only on the Z flag which <u>is</u> affected by a load or store.

The second bug is in the C library rather than the compiler.

Try the following program:

```
#include <stdio.h>

main()
{
    double value;

    pffinit(); /* tell the linker that we need the
printf float routines */

    while(scanf("%F",&value) == 1) /* read a double
from stdin */
        printf("%f\n",value); /* print the value */
}
```

Compile and run the program. When it is waiting for your input type:

```
-0
```

When I did this the program printed:

```
0.0
```

The reason is that _dneg() (in cfloats - called to negate a floating point value) does not check for 0 before changing the sign bit of the value. The original code in _dneg() is:

```
_dneg: ldd ,x    first two bytes of mantissa
 eora #$80       reverse sign bit
 lbra finish     move double to destination
```

If you have the source code for cfloats you can fix it with the following code:

```
_dneg: ldd ,x    first two bytes of mantissa
 tst 7,x         test for exponent of zero
 beq iszero      if so, don't change sign
 eora #$80       reverse sign bit
iszero lbra finish  move double to
*                   destination
```

The fastest way to get the source code for cfloats is to do the following:

Type the following assembly language program:

```
psect find_cfloats,0,0,0,0,0
main: lbra _dnorm
endsect
```

Assemble this program, then link it with the -sm option (symbol map and psect addresses) with output to a disk file or printer for future reference. Do not try to run the program!

Disassemble the program and separate the cfloats portion of the disassembly. (the link map will show its location relative to the start of the program)

Put this line at the beginning of the disassembly:

```
psect cfloats_a,0,0,0,0,0
```

and this line at the end:

```
endsect
```

Any ",Y" references in cfloats refer to "_flacc,y" and jumps outside cfloats refer to _rpterr. (You can rdump your C library with the -a option to verify this)

Some of the "_flacc" references may refer to "_flacc+2,y" or some other constant plus _flacc. Just look for the lowest index constant off Y. (For example, if you see references to "200,y 202,y 204,y 200,y" you should change them to "_flacc,y _flacc+2,y _flacc+4,y _flacc,y",

If you have separated your C library into its component psects you can verify that you have done a correct disassembly with the following batch file:

```
c.asm cfloats.a -o=cfloats.r
* assuming you have named the disassembly cfloats.a
* don't do this in the same data directory as the
* original cfloats.r or you will overwrite it!
cmp cfloats.r /d1/lib/cfloats.r
* assuming the original cfloats library file is
* in /d1/lib
```

If cmp shows that only the date and (maybe) edition bytes have changed (relative locations 00000007 through 0000000C) then you have finished the disassembly and can correct the bug in _dneg().

Now go ahead and delete the program you used to get the disassembly of cfloats so you won't accidentally run it and crash your machine!

Datalight C (for MS-DOS machines) also has the -0. problem. It was a little harder tracking down this bug on the MS-DOS machine (a Tandy 2000 at work) because printf would hang up when it tried to print a -0.! I had to keep resetting the 2000 and inserting print statements in a test program in order to track down the problem on that machine.

It will be interesting to see if these problems have been fixed in the OS-9 68K C compiler.

Sincerely,

Calvin Dodge

Calvin Dodge

P. S. I'm very impressed with your short delay in printing letters. Last year I wrote a letter to you about DIACK Grounded and was surprised to see it in the next issue of '68 Micro. In the middle of October I wrote a letter to BYTE magazine. They just published it in their March issue.

P. P. S. If you decide to print this letter please feel free to edit it (it is rather long). That includes removing the P. S.'s.

SWTPC '86 Dealer 'fest

SWTPC held their 5th annual dealers meeting on the 16 to 19 January '86, in San Antonio, Texas. There were about 25 or so dealers in attendance. Most were new faces but there were several 'old hands' attending, including Joel Heckman of Universal Data Research, Bob Silberman of Audio-Vue, Inc., Franz Fortuny of Centro Cibernetico de Yucatan and Max Wynter of Island Micro System, Inc., and of course, myself from Data-Comp of CPI.

I have attended all of the these meeting in the past (none was held in '85), and this one was very enjoyable. More like a 'reunion' than a meeting. It was good to see some old friends and meet some new ones.

I wish to personally thank Dan Meyer for the kindness and courtesy shown my wife Joyce and myself. Of all of the get-togethers of this type, this one was one of the most enjoyable. It was certainly a very pleasant experience to attend and participate. I hope your have many more - SWTPC.

The meeting this year was very smoothly run. A lot of credit goes to Oscar Rodriguez, Clause Wagner, Lucy Abbott, Steve Fraser, Sue Roland and Susan Shelbourn, of SWTPC. Joe Deres and Tom Stewart of SWTPC engineering were on hand for technical advice and general information concerning new and old SWTPC line products.

Speaking of new products, there were two that caught my attention. One you saw advertised here in 68 Micro Journal, in the past several months, CAD. The CAD (computer aided drafting) was impressive in the quality of the software demonstrated. The plotter demos were also impressive, however, the slowness of the screen updating on the CRT was sorta a drag. The terminal used is their X12 series CRT terminal. Using X Y cursor positioning, it is slower than a memory mapped system. Seems like it would be very simple to implement a high-density video board, just for the CAD system, and pump it out of one of the ports to a color monitor. Now that would be a whizzer! The software is priced about the $2,000.00 level. However, for a complete system, it is competitive.

The other new product is their 68010 VME system. Based on the Motorola VME bus and protocol, it will be priced about in the middle of presently available VME 68010 systems, from other vendors. The VME system can be outfitted with a 68020 board, putting it in the $20,000.00 price range.

The VME system was running both UniFLEX and UNIX system V. The UniFLEX looked as if it needed some finishing touches and the UNIX was just recently ported and lacked some features. I expect that most of these will be attended to by the time this gets into print. As I understood it, either UNIX or UniFLEX, will be shipped, buyers choice.

The remainder of the systems are carryovers from previous years. However, there has been some price adjustments. Also SWTPC is now supporting FLEX again, with their S600 system. I know some of you will appreciate that.

From all indications I received, at the meeting, it appears that SWTPC is expanding into new frontiers. Their newer products and policies seem to be directed toward a more general market. As one old friend to another, we wish them well.

Other new policies from SWTPC were dealer area allocations and product warranties. Seems that each dealer will be assigned a specific area for sales and service. Also the warranty program is to be updated to include better warranty protection for dealers and end-users alike. While the specifics were not really laid out, there were promises that it would be all inked soon. I know many of those attending were looking forward to the details of these programs. All this should make SWTPC products stronger contenders in todays market-place.

Presentation

Over the four days the following presentations were given. Some new products were introduced, and some older ones were spruced up with new wrappings and some inner modifications.

One thing that caught my attention, was that most all serious dealers/programmers, were using Sculptor as the HHL of choice. (Note: Sculptor prices: S.E. MEDIA ads, this issue), and Sculptor(+) articles,recently. Sculptor has come a long way since I first saw it demonstrated. It is now a very popular item, as S.E. MEDIA reports that 1 out of 5 sales are now for 'other-side' systems (IBM, Altos, etc.). The portability of source code is very simple. In our office we develope all our source on a UniFLEX system, then port it to OS-9, IBM, Altos, etc., as the need arises. We find that much of our 68XXX software, written in Sculptor, ports over very easy. And the availability of Sculptor on all those other 'other-side' systems, has opened an entire new market for us. And should for any other S50 dealer/programmer, that is interested in staying afloat, and expanding his market, for software products that were once exclusively for the 68XX(X) systems.

Oscar Rodriguez presented an overview of SWTPC New Marketing Strategy, New Organization, New Products, Service & Warranty polices and sat in on many of the other presentations, fielding questions by the hundreds.

It was my impression at this meeting, more straight answers were given, than at any other I have attended, anywhere. However, some old sores need to be addressed. All in all, it speaks well for their new policies.

Clause Wagner (from next door), their new sales manager served as the dealer-SWTPC interface. Klause is a born salesman. A very likeable chap, and could sell hogs hair shampoo. His main-liner was on Friday the 17th, when he gave a resounding sales/marketing course. When it was over I dashed out and even attempted to sell the waiter a CAD system. No sale, seems one of the other fellows beat me there. Seriously though, it was something that many appreciated and needed.

Susan Shelbourn gave a presentation on Dynacalc. While most of the old hands knew it already. It was well received by the newer dealers. Joe Turner could be proud.

Steve Fraser presented UniScript for UniFLEX. However, most there had already tried it and were not too impressed. Most all complained that it still needed some fixing up. Funny, no one (except me) gave a plug for STYLO. I personally would rather (and do) use STYLO than any of the other processors presented. TSC also has a newer processor for their 68000 systems, but I have not tried it yet, so can tell you little. It was not presented.

Steve also gave presentations on SWTPC software, Sculptor and SWTPC service aids.

Sue Rowland was there with the CAD system. Giving both verbal presentations and hands on experience (at the plant) with the CAD system. We will be doing a more indepth review of the documentation of this system, in the months to come.

Also in the CAD department, Mike Almeguer told us about the plotter ramifications on the CAD system.

Peter Hite of Technical Training and Consulting presented their 3d party POS (point of sale) interface system.

Dr. David Truemper of Three Pros Computers was there telling us about their Church Records Management System, This one we are familiar with as our Data-Comp division has sold this package running on the 68020 with UniFLEX, to one of the largest churches in the southeast. It is nicely done and because it is in Sculptor, it will port nicely to other systems.

Werner Schoratein of EDV Systeme & Peripherie, W. Germany, gave an overview of their Network & File Transfer System. This is a type of network communications package and got a lot of interest. However, it is not completely in final form, and some specifics were not available.

Ken Mikesell of Univall (a SWTPC company in the POS and cash register market) gave a talk and answered questions on their products. The presentation and support material for these products were professional and tasteful. If I were a cash register sales type, this would certainly get a lot of my attention.

Joel Heckman of Universal Data Research was impressive with the array of BASIC and Sculptor based software they have developed. Everything from municipal management to general business packages. We (and I think a lot of others) did not realize that they had all that good stuff.

Also you might have guessed it, I got my two cents worth in. So, all in all, I consider it one of the better SWTPC dealer meetings and reunion.

I know that I have left someone, deserving mention, out. For that I offer my apology. However, I want to say again it was very enjoyable and enlightening. I hope next year there is another one. And I certainly wish SWTPC good luck in the months and years to come. After all, it is nice to be associated with the 'Oldest Microcomputer manufacturer, in the WORLD!'

DMW

MOTOROLA ANNOUNCES 20 MHZ MC68020

AUSTIN, TEXAS, JANUARY 30, 1986.... Motorola Microprocessor Products Group announces the 20 Megahertz (50 nanosecond) version of the MC68020 32-bit microprocessor. High quality design and production efforts contribute to Motorola's ability to accomplish such increased performance.

Introduced in mid 1984, the MC68020 is the first commercially available complete 32-bit MPU with 32-bit nonmultiplexed address and data buses. With 3 - 4 MIPS performance and burst speeds to 10 MIPS, The MC68020 is the 32-bit performance standard. An enhanced instruction set and instruction cache are included on the MC68020 for high-speed performance.

The 20 MHz MC68020 is being sampled now, with production quantities scheduled for 2Q 86. The 20 MHz MC68020 is priced at $771 in 100 piece quantities. Production quantities of 12.5 MHz and 16.67 MHz MC68020s are currently available.

MOTOROLA ANNOUNCES "END-OF-LIFE" ORDERS ON MANY DATA CONVERSION PRODUCTS

Phoenix, Arizona, February 17, 1986 ... Motorola has announced they are now taking "end-of-life" orders on a large number of older data conversion circuitry. The company is phasing out these components as part of a strategic shift of resources to thir new line of video-speed data conversion devices based on thir MOSAIC process. The current Motorola data convereion portfolio includes video-speed flash and non-flash converters, commodity D/A and A/D converters, tele-communication encryption devices, a family of precision 10-12 bit D/A converters and various other conversion circuits.

The following parts will be phased out over the next two years. The dates cited for orders and deliveries are the final dates that new or increased orders can be accepted:

PRODUCT	DESCRIPTION	ORDERS BY	DELIVERY BY	NOTES
MC1405	A/D SUBSYSTEM	3/31/86	6/30/86	EXTENDED TO '86
MC1550G	RF/IF AMPLIFIER	12/31/86	12/31/87	USE MC1590/1350
MC2510	10-BIT DAC	12/31/86	12/31/87	----------
MC3410,C	10-BIT DAC	12/31/86	12/31/87	----------
MC3512	12-BIT DAC	12/31/86	12/31/87	----------
MC3412	12-BIT DAC	12/31/86	12/31/87	----------
MC6890	8-BIT MPU DAC	12/31/86	12/31/87	----------
MC10319L	7-BIT ADC	12/31/86	12/31/87	USE MC10319
MC10321L	7-BIT ADC	12/31/86	12/31/87	USE MC10319
MC1506L	6-BIT MULT.DAC	12/31/86	12/31/87	----------
MC1406L	6-BIT MULT.DAC	12/31/86	12/31/87	----------
AD562	12-BIT DAC	4/30/86	12/31/86	ANALOG DEVICES
AD563	12-BIT DAC	4/30/86	12/31/86	ANALOG DEVICES

MOTOROLA DISCONTINUES FOUR 8-BIT PARTS

AUSTIN, TEXAS, March 13, 1986... Motorola Microprocessor Products Group is discontinuing the manufacture of all packaged version of the following parts: MC6803E, MC6805P4, XC68120, and XC68121. They will be available for lifetime buys for a period of two years with delivery scheduled within three years.

The MC6803E is a high-density 8-bit HMOS microprocessor. The part is designed for use in systems where the internal clock needs to be synchronized with clocks of external parts or systems. Except for the on-chip ROM and oscillator, this chip has all the features of the MC6801. These features include 128 bytes of RAM, parallel I/O lines, a serial interface, and a three-function programmable timer. This 40-pin part is available in plastic and ceramic packages and is compatible with the M6800 family.

The MC6805P4 is a low-cost 8-bit microcomputer with on-board ROM and RAM, an efficient instruction set, clock, and timer. It has zero-crossing detection and implements a self-test program. This 28-pin part is available in plastic, cerdip, and ceramic packages.

The XC68120 and XC68121 are ceramic sided-brazed packaged general purpose Intelligent Peripheral Controllers (IPCs) which serve as interfaces between a peripheral and an M6800 orM68000 family microprocessor. On-chip dual-port RAM and 21 I/O lines make these 48-pin parts useful in many applications. The XC68120 has 2K bytes of on-chip ROM allowing it to use all addressing modes, but the XC68121 supports only expanded addressing modes since it has no ROM.

MOTOROLA DISCONTINUES EIGHT 8-BIT CHIPS

AUSTIN, TEXAS, March 13, 1986... Motorola Microprocessor Products Group is discontinuing the manufacture of all packaged versions of the following parts: MC6822, XC6829, MC6835, MC6847T1, MC6847YP, MC6808, and MC6845R1. They will be available for lifetime buys at closeout prices for one year with delivery scheduled within two years.

The MC6822 is an Industrial Interface Adapter (IIA) that is used to interface peripheral equipment to the M6800 family of microprocessors through two 8-bit bidirectional data buses and four control lines. The device's configuration is selected upon system initialization by the MPU, and no external hardware is required. The 40-pin chip is available in plastic, ceramic dip, and ceraminc side-braze packages.

The XC6829 is a Memory Management Unit (MMU) principally used to expand the memory of the MC6809 from 64K bytes up to 2M bytes. Each MMU can handle up to four tasks concurrently. The 40-pin part is available in plastic, ceramic dip, and ceramic side-braze packages.

The MC6835 is a ROM-based CRT Controller which links an MPU system to a raster scan CTR display. Designed for flexibility, this part can function with stand-alone terminals or with clusters and is ideal for video game applications. The 40-pin part is available in plastic, ceramic dip, and ceramic side-braze packages.

The MC6839 is an n-channel silicon gate Floating Point ROM which provides a simple and reliable solution to many numerial processing applications and satisfies the IEEE Standard for Binary Floating Point Arithmetic Draft 8.0. Used in multitasking and real-time applications, this part is position independent and offers single and double percision. Extensive error-checking and exception signalling make this a versatile tool in MC6809 systems. The 24-pin part is available in both ceramic side-braze and plastic packages.

The MC6846 is a ROM-I/O Timer and makes it easy to develop a 2-chip microcomputer system when used with an MC6802. It has 2K bytes of mak-programmable ROM, an 8-bit bidirectional data path with additional control lines, and a 16-bit programmable timer-counter. This part can interface with the entire M6800 family with no extra logic in most cases. The 40-pin part is available in plastic, ceramic dip, and ceramic side-braze packages.

The MC6847YP is a plastic package interlaced Video Display Generator (VDG) part. For use in applications such as video games, process control displays, home and educational computers, ad graphics, this chip interfaces M6800 microprocessors to black and white NTSC television receivers. The 40-pin part can be connected to a television set using the MC1372 TV Chroma and Video Modulator.

The MC6847T1 is an enhanced vrsion of the MC6847 non-interlaced Video Display Generator. All address lines except the least significant bit have been

replaced by an 8-bit I/O port, as the need for external interface circuitry is reduced. The 40-pon part is available in plastic, ceramic dip, and ceramic side-braze packages.

The MC6808 is an n-channel silicon gate MPU similar to the MC6800 but possesses an internal clock as well. This part is software compatible with the entire M6800 family and suppoirts 6 -bit addressing. The 40-pin part is avilable in ceramic side-braze and plastic packages. Recommened replacement is the MC6802 which is pin-for-pin compatible.

The MC6845R1 is a CRT Controller (CRTC) which performs the interface between a raster-scan CRT and a microprocessor. The chip is flexible, and all terminal functions are under MPU control. The CRTC provides refresh memory addressing and video timing fuoctions and is useful in single-color or multi-color CRT applications. The 40-pin part is available in plastic, ceramic dip, and ceramic side-braze packages.

GIMIX INC.

1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609

(312) 927-5510 • TWX 910-221-4055

GIMIX will be exhibiting at NCC 86 in Las Vegas from June 16th-19th at Booth No. 4235. We invite any of your subacribers who will be attending to visit us. We will have a limited amount of complimentary guest tickets available. Please contact Richard Don if you need one.

Regards

Richard Don

GIMIX INC
Richard Don

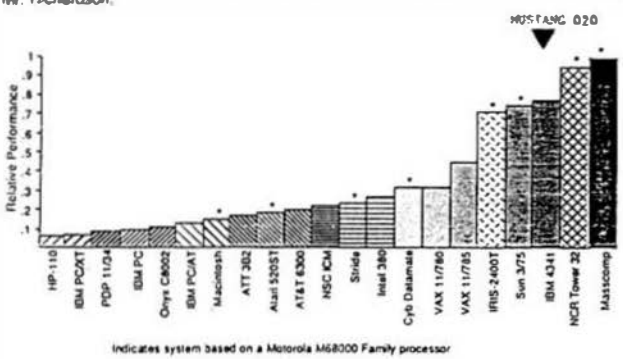
M68000 MICRO MINUTES

MC68020 PROVES ITS MAINFRAME PERFORMANCE ON DHRYSTONE

A recent "USEnet" posting by Rick Richardson of PC Research, Inc. which compared the performance of 145 different machines — from a Commodore 64 to an Amdahl 5860 — proves conclusively that the MC68020 can deliver mainframe performance for a micro price.

The Dhrystone benchmark, written by Rheinhold Weicker, is a series of routines which, when taken as a whole, contain a statistically correct distribution of the various instructions, addressing modes, and (non-floating point) data types normally found in the "average" industrial/business program. The benchmark, which was originally written in Ada with guidelines for proper conversion to both C and Pascal, has been widely used to compare the non-floating point performance of various machines in the same way the Whetstone benchmark is used to compare floating point/transcendental performance.

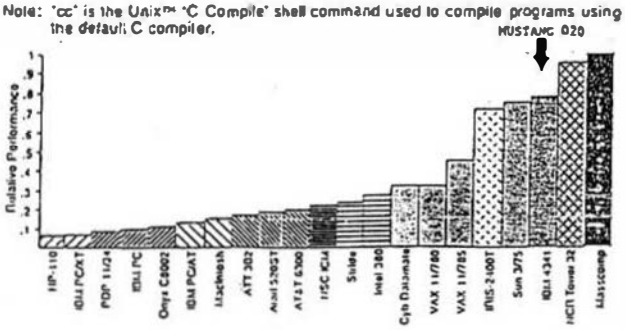
The results presented here are just a sampling of the results listed on the USEnet by Mr. Richardson.



Indicates system based on a Motorola M68000 Family processor

Machine Type	Microprocessor	Operating System	Compiler	Dhrystones/sec No Regs	Regs
HP-110	8086 - 5.33MHz	MSDOS 2.11	Lattice 2.1c	284	284
IBM PC/XT	8088 - 4.77MHz	PCIX	cc	257	287
PDP 11/34	—	Unix™ V7M	cc	357	439
IBM PC	8088 - 4.77MHz	PCDOS 2.1	Aztec C v3.2d	423	458
Onyx C8002	Z8000 - 4MHz	IS/1 1.1 (V7)	cc	476	511
IBM PC/AT	80286 - 6MHz	PCDOS 3.0	CI-C86 2.1	666	684
Macintosh	68000 - 7.7MHz	—	MegaMax C 2.0	661	709
ATT 3B2/300	WE32000 - 7.2MHz	Unix™ 5.0.2	cc	735	806
Atari 520ST	68000 - MHz	TOS	QRS C	839	846
AT&T 6300	8086 - 8MHz	MSDOS 2.11	Aztec C v3.2d	862	943
NSC ICM-3216	NS32016 - 10MHz	Unix™ SVR2	cc	1041	1084
Stride	68000 - 12.5MHz	System V/68	cc	1063	1136
Intel 380	80286 - 8 MHz	Xenix R3.0up1	cc	1250	1316
Cyb Datamate	6 010 - 12.5MHz	Uniplus 5.0	Unisoft cc	1470	1552
VAX 11/780	—	Unix™ 5.2	cc	1515	1562
VAX 11/785	—	Unix™ 4.3bsd	cc	2135	2136
IRIS-2400T	68020 - 16.67MHz	Unix™ Sys V	cc	3105	3401
Sun 3/75	68020 - 16.67MHz	SUN 4.2 V3	cc MUSTANG 020	3333	3571
IBM-4341	Model 12	UTS 5.0	?	3555	3685 MUSTANG
NCR Tower 32	6 020 - 16.67MHz	Sys 5, Rel 2.0	cc	3846	4545
Masscomp 5600	68020 - 16.67MHz	RTU V3.0	cc (4.0)	4504	4746

Note: "cc" is the Unix™ "C Compile" shell command used to compile programs using the default C compiler.



Indicates system based on a Motorola M68000 Family processor

NOTE: Benchmarks for the MUSTANG-020

MUSTANG-020 68020 - 16.67MHz "REGFILE" TSC C 3668 3816

The above is another in the long list of benchmarks we have comparing the MUSTANG-020 and other systems.

Special notice should be given the position of the MUSTANG-020 as compared to systems either side on the chart. And the "Register" and "No Register" times. It is apparent the MUSTANG-020 is far under priced. Systems either side of it, on the charts, sell in the $50,000.00 to $100,000.00 + range. Even after the price increase (next quarter), it will still be the best value in systems of this power and bus size (32 bits). And NOW it is even a BETTER BUY!

MICROWARE®
Microware Systems Corporation
1866 N.W. 114th Street
Des Moines, Iowa 50322

Telephone 515-224-1929
FAX 515-224-1352
Telex 910-520-2535

MICROWARE SYSTEMS CORPORATION'S

COM: COMMUNICATIONS PROGRAM

Description: COM allows communications between an OS-9 based computer system and any remote computer system using a modem or direct hardwired connection. The remote computer does not have to be an OS-9 system. Consequently, COM can be used to interface your OS-9 system to timesharing systems, information utilities and videotex services.

COM works by interchanging data between your terminal and a designated I/O port which is connected to the remote system. COM can transmit text files or Motorola S records from your system to the remote system, or can store incoming data from the remote system on a disk file.

COM is designed to communicate with a remote system via an RS-232 serial port. Both the terminal and communications port must be interrupt-driven, and the terminal used must be able to operate at a baud rate equal to, or faster than, the communications port on the terminal.

COM has two operational modes: Communications Mode and Control Mode. In Communications Mode the usual OS.9 control keys are disabled and the terminal only responds to the control keys of the remote system. All data from either the terminal or remote system is immediately transferred on a character by character basis. Control Mode provides the user with the ability to change operating parameters, upload/download files and access the Shell.

COM also provides a programmable "function key" feature that allows up to ten user-defined text sequences to be stored for transmission via a simple keyboard command. This feature eliminates repetitive typing of commonly used keyboard entries such as a log-on. As this feature is software implemented, no special terminal is required.

Ordering Information:

Manual and software diskette	$125.00
Manual only	$12.00

To place your order, please contact your local Microware Systems representative, or call Microware Systems headquarters at (515) 224-1929.

ROBCON OY

Postiosoite Postadress
P.O.Box 9
SF-00391 HELSINKI
Finland

Käyntiosoite Street adress
Hankasuontie 9
SF-00390 HELSINKI
Finland

Puhelin Telefon
(+ 358 0) 543 144
Telex: 125316 robco fi

ROBCON - Euroka Electronics 32 Bits Single Board VME Super Micro

ROBCON announce the availability of the CPU 020, extra ordinary powerful 68020 based super micro. Comparing features onboard to other commercially available 68020 based products, this new released CPU 020 has clearly more functions than any other 68020 based VME board in the world. ROBCON CPU 020 is based on universal VME-bus, which is the benefit for those who would like to keep independence of their own.

ROBCON CPU 020 modules are designed to be used with VME and MVMX 32 modules. They are suitable for applications requiring large memories, fast computing speed and/or multiprocessing.

The ROBCON CPU 020 is based on the most advanced 32 bit microprocessor of today, the MC 68020. Thus it is capable of being both ordinary bus master, or as a slave module and a system controller module. The new released board meets the VME revision C and IEEE P1014/D1.0 and the MVMX32 bus specifications.

In addition there is onboard, One Mega byte dual port dynamic RAM memory with 310 nS cycle time. The sockets for 64 Kbyte on-board EPROM memory are also included. The addressing range is as large as four Giga byte. Fast VME/MVMX32 access is based on two separate Giga byte address sections.

The data path is internally and externally 32 bits. Maximum bus rate is 20 Mbytes per second. Execution speed is from 2 to 8 million instructions per second (MIPS). Minimum execution time for one instruction is 0 nS! The CPU 020 uses fast separate access in 16 Mbyte VME bus sections for D16/A24, D32/A24 and even for D16/A16 bus transfer types.

The CPU 020 has VME bus priority (PRI) arbiter, four level VME bus requestor with dynamic requesting level and four dynamically selectable bus release modes: release when done (RWD), release on request (ROR), release never (RNE) and release when time out (RTO).

The new released CPU 020 also has the VME bus controller: sysclock, reset and bus time out. There is seven level VME bus interrupt handler. Levels to be served are dynamically selectable.

The module consists two asynchronous serial communication channels with RS-232-C interface (async, bisync or HDLC). The other has additional connector and drivers for RS 422 interface.

The most features and options on-board are conveniently selectable by software instead of jumpers!
Furthermore, there is on-board 68881 IEEE format floating point co-processor with the following features:
- calculation speed 300 K flops in add, substraction takes 2.25 uS, 32 bit multiplication takes 2.5 uS and 32 bit division takes 5.5 uS only
- supports 32, 64 and 80 bit extended IEEE format
- support trigonometric and logarithmic functions
- supports conversions between floating point, integer and binary coded decimal.

In addition ROBCON CPU 020 has 68851 demand page Memory Management Unit.

CERN (European Particle research center) has accepted, after investigations, this module to be used in numbers of different applications.

ROBCON - Euroka Electronics is willing to send information concerning the CPU 020 VME -module and other modules available on request through the office in Helsinki.

Ágúst H. Bjarnason

Electronic Engineer, clu. Ing.

HOLTSBUD 44, 210 GARDABAER, ICELAND

Dear Editor:

Thank you very much for including the Macintosh computer in your magazine.

I did the 128K to 512K memory upgrade according to the instructions in a recent issue, and it took only about 3 hours and saved me several hundred dollars.

Do you, or any of your readers, know of any amateur radio software (morse, radioteletype etc.) for the Mac? I would appreciate any information very much.

Best Regards,

Ágúst H. Bjarnason

Ágúst H. Bjarnason, TF 3 OM
Holtsbud 44, 210 Gardabaer
Iceland

with C Source

$229

*** **PAT** from **Southeast Media** -- A full Feature screen oriented **TEXT EDITOR** with all the best of "PIE". For those who swore by and loved only PIE, this is for you! **All PIE features and much more!** Too many features to list. And if you don't like these, change or add your own. C source furnished. Easily configured to your CRT, with special config section.

68008 - 68000 - 68010 - 68020 OS-9 68K Special: $229.00
Includes full C source

68008 - 68000
COMBO - PAT JUST $249
68010 - 68020

*** **JUST** from **Southeast Media** -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the **Display.** **"User Configurable"** for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with **PAT** or any other editor. The ONLY stand alone text processor for the 68XXX available, and at a very low price

68008 - 68000 - 68010 - 68020 OS-9 68K Special: $79.95
Includes full C source!

$79.95

with C Source

Classified Advertising

68008 HARD DISK SYSTEM - COMPLETE

512K 68008 system, 10 megabyte hard disk, Xebec 1410A HD controller, 80 track double side, double density floppy. Complete with cabinet/power supply. Taken in on Mustang-020 trade-in. Version 1.2 OS-9, Basic09, Stylo, Mail Merge, Spelling Checker, Dynacalc - like new - original price $2,900.00 range (advertised) - **SPECIAL - ONLY $1750.00.**
615 842-4600 - Data Comp, ask for Don or Tom.

Winchester 10 Megabyte Drives
Two (2) 10 Megabyte Hard-Disk Winchester Drives. Working - were removed for upgrade to larger drives.

1 - RMS Model #509	$275.00
1 - Seagate Model #412	$275.00

(615) 842-4600 Tom 9-5 EST.

LSI 68008 CPU card, "C" Compiler and Digital Research CPM/68K $350.

Tano Outpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $595.

MICROKEY Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p. 51 DEC. 84 68' Micro Journal. $1800.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS. $745.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $359.00 ready to run.

S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1290.

1-CDS1 20 Meg Hard Disk System with Controller $1000.

(615) 842-4600 M-F 9 AM to 5 PM EST

***

Several 2MHZ 64K Static Ram cards, $75 each. Misc. SS50 and SS30 cards. Call with your needs.
(703) 273-6629 evenings.

***

Sage IV multi-user micro, 10 mHZ 68000 cpu, 1 Mb RAM, 18 Mb Winchester, 2 DSDD-80 floppies, Freedom 100 Terminal, UCSD p-System with word processor & spreadsheet, Volition Systems Modula-2 with ASE, Whitesmith's IDRIS (Unix look-alike) with full C.
$4,500.00 (309) 246-8164

***

WANTED blank Data Systems 68 boards. Alen Gordon, M.D. 1435 W. 49th Place, Hialeah FL 33012 (305) 822-1100

***

ATTENTION SWTP Users S/09 Upgradeable to S+ 64K Memory, 8" Floppy, 10MB Winchester, 6 Serial Ports, 1 Parallel Port, Software, Manuals, Qume Terminal. MAKE OFFER!
Call (314) 469-3100

***

Smoke Signal Chieftain, 20 Mg. Winchester, 20 Mg. tape, DSDD floppy, 1 Mg. memory, 4 serial & 2 parallel ports, software and manuals. Some repairs needed. $6,000 or best offer.
(212) 288-3614

***

For Sale: Two Qume 8" Drives - Brand new unopened in original packaging- for 6800 System - 414-657-9390

***

For Sale: 6800 Computer with smoke signal boards - 2- 8" Drives 2-Soroc terminals working order- Database + Word Processing Software- Full Documentation
414-632-5151

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4600
For Ordering
Telex 5106006630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, I2716, T2516, I2732/2732A, MCM68766/6, I2766/2766A, T2564, I27128/27128A, and I27256. (=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's Intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l RETURN ..... to the operating system.
l. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (IREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BIT10, HARDIO, HEXIO, FLEXIO, SCIPACK, SFRSUBS, BASTRING, and REALCON.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V11 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independantly or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker - Serial poll - Single or Dual Primary Address
  - Listener - Parallel Poll - Secondary Address
  - System Controller - Group Trigger - Talk only ... listen only
- Fully documented with a complete reprint of the HELOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	S5-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEx or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES EXCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order. VISA or MASTER-CARD also accepted.

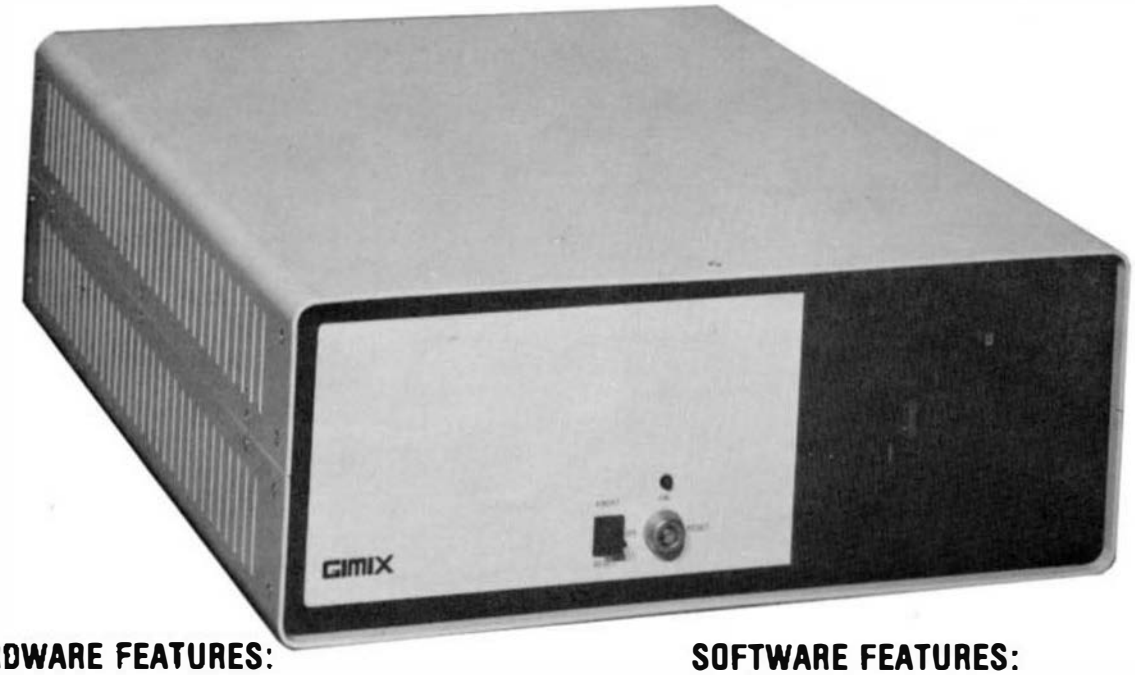
WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

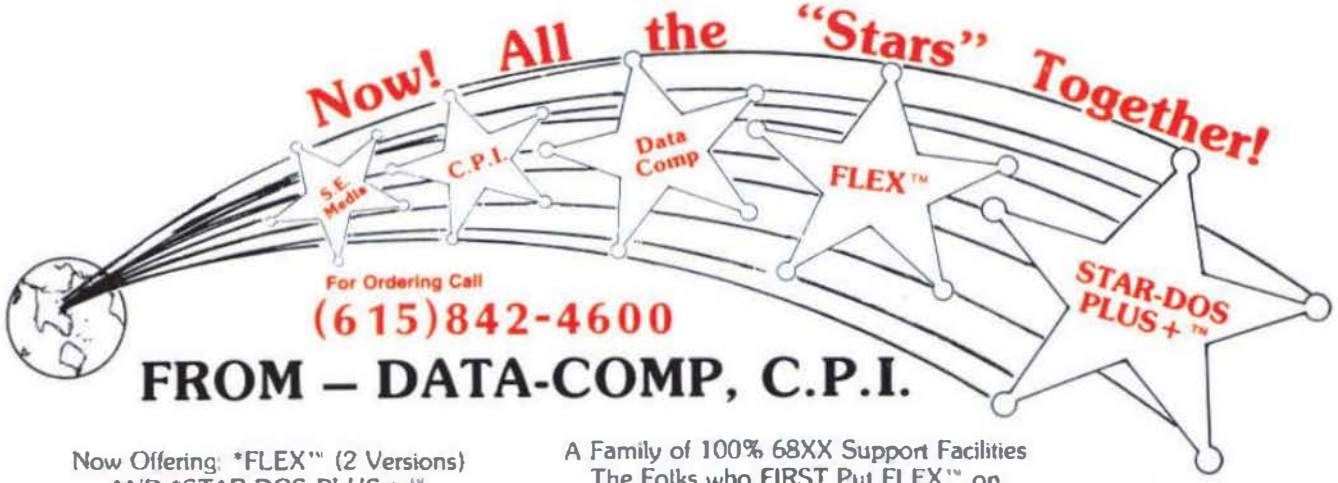
- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS. $469.95

* Specify What CONTROLLER You Want J&M, or RADIO SHACK

THINLINE DOUBLE SIDED
DOUBLE DENSITY 40 TRACKS $129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,O,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K RXPAND TO 128K	$169.95
EPSON MX-MX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES	CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

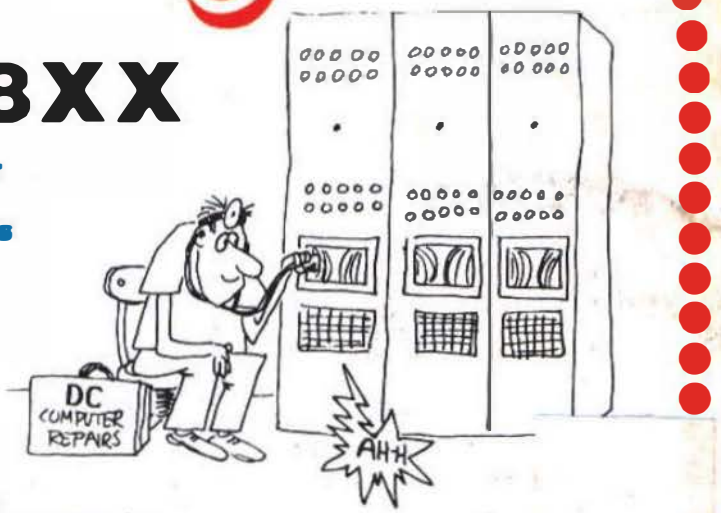
S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS



NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users, including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and confirm a **Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - YET, WE DO NOT HAVE EVERYTHING! But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

This ↑

Not Th ↓

000-22 A/E MJ
MR. MICKEY FERGUSON
P. O. BOX 87
KINGSTON SPRINGS TN 37082

DATA-COMP

5900 Cassandra Smith Rd.
Hixson, TN 37343

☎ **(615)842-4607**

Telex 5106006630